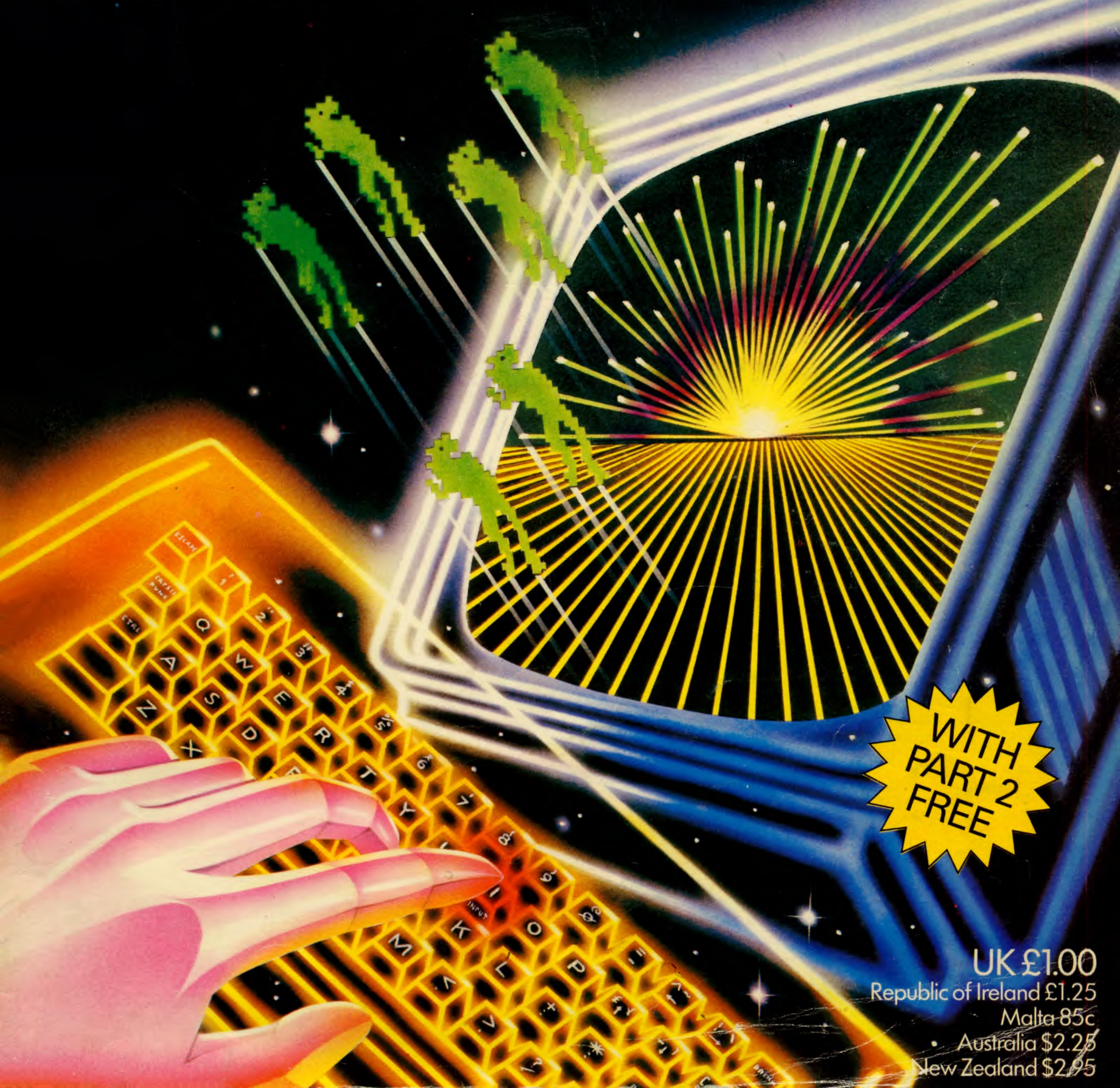
A MARSHALL CAVENDISH **1** COMPUTER COURSE IN WEEKLY PARTS

INPUT

LEARN PROGRAMMING – FOR FUN AND THE FUTURE



WITH PART 2 FREE

UK £1.00
Republic of Ireland £1.25
Malta 85c
Australia $2.25
New Zealand $2.95

INPUT

Vol 1 No 1

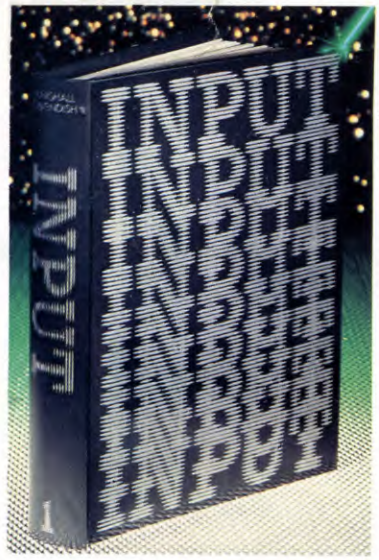
INDEX

The last part of INPUT, Part 52, will contain a complete, cross-referenced index. For easy access to your growing collection, a cumulative index to the contents of each issue is contained on the inside back cover.

PICTURE CREDITS

Front cover: Ian Taylor. Inside front cover: Binder picture, Graeme Harris/AA Page/Mt. Tamborine Observatory. Pages 8, 9, 12, 13, 15, Jeremy Gower. Pages 16-21, illustrations, Peter Bentley; photos, John Darling. Pages 22-23, (top), Kevin O'Keefe, (bottom), Chris Lyon. Pages 24-25, Kevin O'Keefe. Page 27, illustration, Chris Lyon; photo NASA. Pages 28-29, illustration, Chris Lyon; photo, Tony Stone Associates. Pages 30-31, illustration, Chris Lyon; photo, Jerry Young. Page 32, illustration, Chris Lyon; photo, Rex Features Ltd. Computer effects by J.D. Audio Visual.

Published by Marshall Cavendish Partworks Ltd, 58 Old Compton Street, London W1V 5PA, England. Printed by Artisan Presss, Leicester and Howard Hunt Litho, London.



There are four binders each holding 13 issues.

HOW TO ORDER YOUR BINDERS

UK and Republic of Ireland: Send £4.95 (inc p & p) (IR£5.45) for each binder to the address below:
Marshall Cavendish Services Ltd, Department 980, Newtown Road, Hove, Sussex BN3 7DN

Australia: See inserts for details, or write to INPUT, Gordon and Gotch Ltd, PO Box 213, Alexandria, NSW 2015

New Zealand: See inserts for details, or write to INPUT, Gordon and Gotch (NZ) Ltd, PO Box 1595, Wellington

Malta: Binders are available from local newsagents.

BACK NUMBERS

Copies of any part of INPUT can be obtained from the following addresses at the regular cover price, with no extra charge for postage and packing:

UK and Republic of Ireland:
INPUT, Dept AN, Marshall Cavendish Services, Newtown Road, Hove BN3 7DN

Australia, New Zealand and Malta:
Back numbers are available through your local newsagent

COPIES BY POST

Our Subscription Department can supply your copies direct to you regularly at £1.00 each. For example the cost of 26 issues is £26.00; for any other quantity simply multiply the number of issues required by £1.00. These rates apply anywhere in the world. Send your order, with payment to:
Subscription Department, Marshall Cavendish Services Ltd, Newtown Road, Hove, Sussex BN3 7DN
Please state the title of the publication and the part from which you wish to start.

HOW TO PAY: Readers in UK and Republic of Ireland: All cheques or postal orders for binders, back numbers and copies by post should be made payable to:
Marshall Cavendish Partworks Ltd.

QUERIES: When writing in, please give the make and model of your computer, as well as the Part No., page and line where the program is rejected or where it does not work. We can only answer specific queries – and please do not telephone. Send your queries to INPUT Queries, Marshall Cavendish Partworks Ltd, 58 Old Compton Street, London W1V 5PA.

INPUT IS SPECIALLY DESIGNED FOR:

The SINCLAIR ZX SPECTRUM (16K, 48K, 128 and +), COMMODORE 64 and 128, ACORN ELECTRON, BBC B and B+, and the DRAGON 32 and 64.

In addition, many of the programs and explanations are also suitable for the SINCLAIR ZX81, COMMODORE VIC 20, and TANDY COLOUR COMPUTER in 32K with extended BASIC. Programs and text which are specifically for particular machines are indicated by the following symbols:

SPECTRUM 16K, 48K, 128, and +

COMMODORE 64 and 128

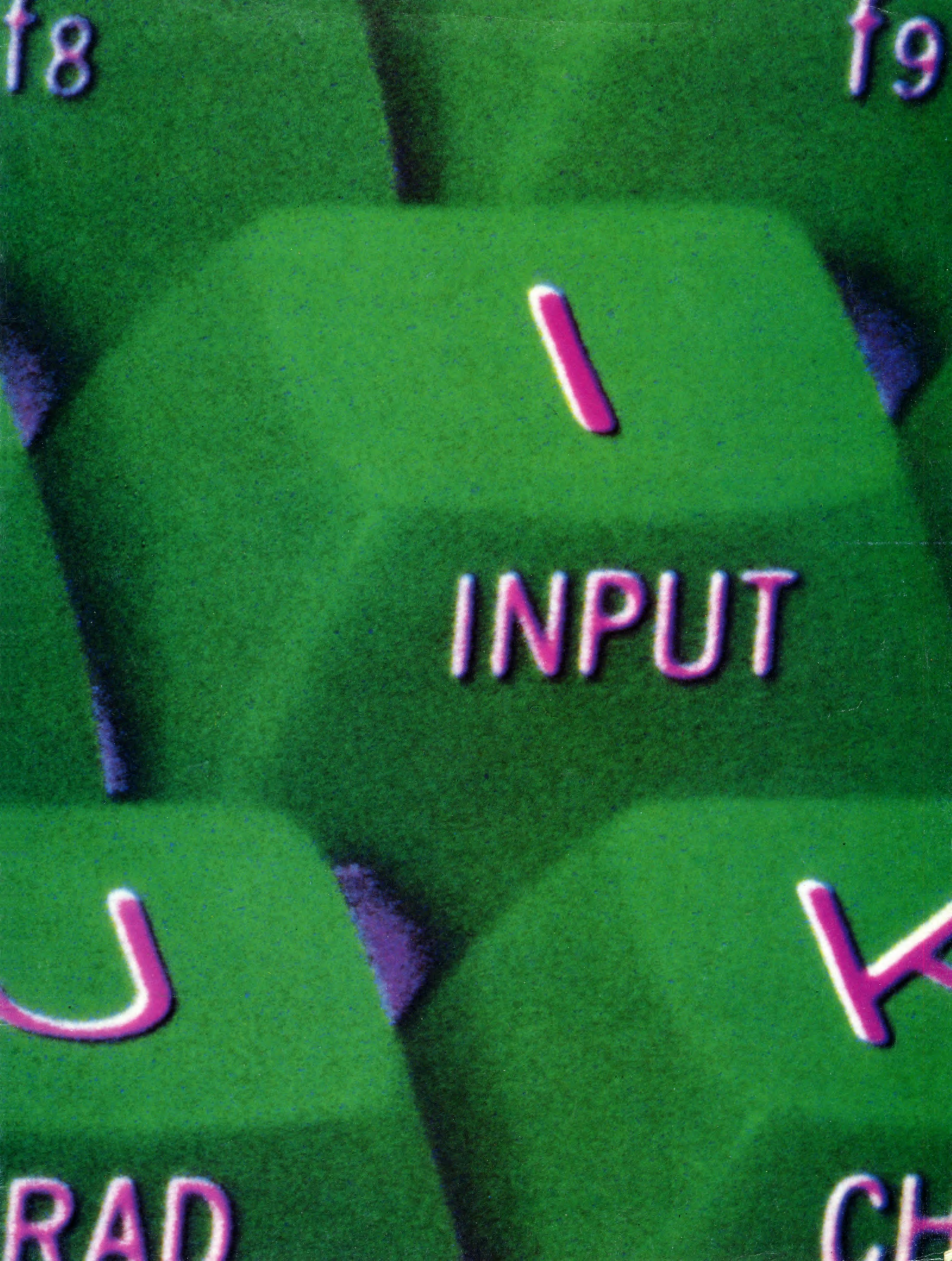
ACORN ELECTRON, BBC B and B+

DRAGON 32 and 64

ZX81

VIC 20

TANDY TRS80 COLOUR COMPUTER


f8
f9
I
INPUT
U
K
RAD

THINK OF A NUMBER... ANY NUMBER

Dice throwing, quizzes and many other computer games use random numbers to create a seemingly irregular sequence of events. Here we show you how

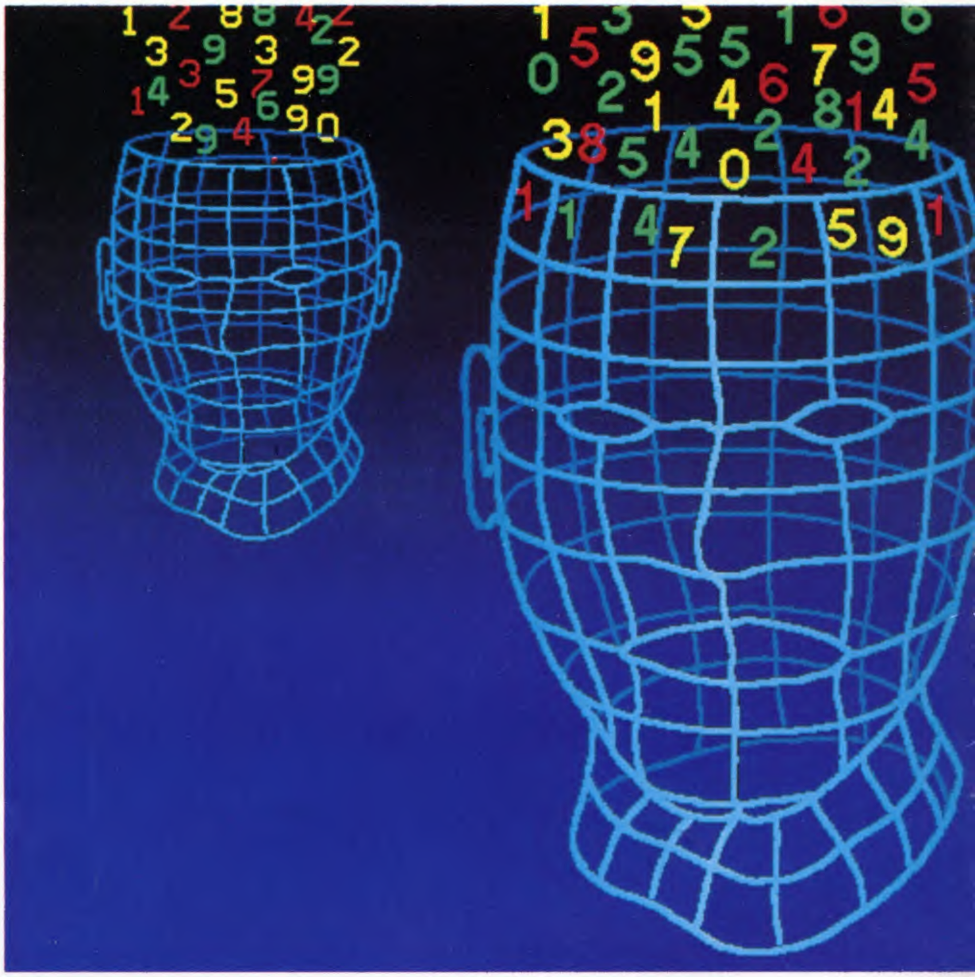
Learning to program a computer is a bit like learning to play football.

Theoretically, you could learn football by practising one skill at a time, stepping out onto the pitch for an actual game only when you have mastered every move. But you would learn slowly — and have very little fun in the process.

Similarly with programming. One way is to plough your way through a manual which teaches one function at a time — but never how to combine them. A better way is to 'step out on the pitch' and start playing! So where do you begin?

GUESSING THE NUMBER

The easiest of all home computer games to program is the one in which the computer 'invents' a random number and the player tries to guess what it is.



Q+A

Some computers use the keyword RANDOMIZE. What exactly does this do?

The Spectrum uses the function RANDOMIZE 1 (or any other number) to make sure that a program repeats the same sequence of random numbers each time it is run. This is useful if you are trying to debug a program as it is easier to find any mistakes if the program does the same thing each time.

The Acorn, Commodore and Dragon computers use RND(−1) for the same purpose. Again, any number will do but it must be negative.

The Spectrum can also use the function RANDOMIZE without a number, or with a Ø after it, and this has quite the opposite effect. It makes the random number sequence even more random.

THE RND FUNCTION

All home computers have a *random number generator* to allow you to invent such games. This is operated in BASIC by the function RND.

On some computers the numbers it produces, however, are not much use in their original form – they are all decimalized fractions between Ø and Ø·99999999.

To demonstrate this, type in this program, first entering NEW to clear out any existing program:

S

```
1Ø LET X=RND
2Ø PRINT X
3Ø GOTO 1Ø
```

Acorn

```
1Ø LET X=RND (Ø)
2Ø PRINT X
3Ø GOTO 1Ø
```

Commodore 64, Vic 20, Dragon

```
1Ø LET X=RND (1)
2Ø PRINT X
3Ø GOTO 1Ø
```

(Remember to tap ENTER or RETURN — whichever your computer has — after each line of program.)

When you RUN this program, all you will get will be a string of long, decimalized fractions — far too hard for a guessing game (for why this is, see the chart on page 7.)

So how can you get the computer to produce only whole numbers? By putting INT (short for *integer*, or whole number) in front of the RND, thus:

- THE **RND** FUNCTION
- HOW RANDOM NUMBERS ARE CREATED
- USING VARIABLES
- THE **INPUT** STATEMENT
- ADDING **IF . . . THEN** FOR COMPARISONS
- TWO NUMBERS GAMES YOU CAN PROGRAM
- RANDOM NUMBER RANGES



```
10 LET X=INT (RND*6)
```

```
10 LET X=INT (RND(1)*6)
```

This generates a whole number between 0 and 5. On other computers you do not have to use INT. To generate whole numbers between 0 and 5, for example, you just type

```
10 LET X=RND (6)-1
20 PRINT X
30 GOTO 10
```

(Remember to tap ENTER or RETURN after each line.)

Whichever computer you are using, you are not limited to using a range of numbers from 0 to 5. You could equally well choose 10, or even 10,000 — but this last would make the guessing game very hard indeed!

USING VARIABLES

In writing the short program above, you have not only selected a random number but also given it a name: X. From now on, whenever X crops up, the computer will know you mean 'that random number you just thought of.'

Such a name, which allows the computer to identify a number — so that it can compare it with another number, or multiply it, or divide it, or whatever you wish — is called a *variable*.

THE INPUT STATEMENT

Having generated the random number, the next step is to warn the computer to accept your guess at what that number is. So you use the INPUT statement. This tells the computer to wait until some information has been typed in.

INPUT by itself, however, means nothing. The computer must have a name by which it can identify the data it will receive. So, as before, a variable is used. In this case it is G (for 'guess'), but it doesn't have to be — it could be any other letter, or combination of letters, or even the whole word GUESS if you feel that energetic.

So the whole statement is as follows (but do not type it in yet):

```
INPUT G
```

Now that your computer knows the secret number it has generated, and your guess, it can compare the two. This is done in a very similar way to ordinary English:

```
IF X=G THEN PRINT "WELL DONE"
```

The IF...THEN statement is clearly a very useful one. In programming, you will use it over and over again.



1. The cursor — a small underline on this machine, but a 'blob' on others — shows where the next character you enter will appear on the screen of the television set

Some computers, such as the Acorns and Dragon, go one better. They have an extension of the statement, IF...THEN.... ELSE. In this case the computer does something else if the two numbers are *not* equal.

On those computers without the ELSE extension, such as the Sinclair and Commodore home computers, the computer automatically skips to the next line of program if, in this case, the two numbers are not equal.

PROGRAM OUTLINE

Now type in this program:
(The < > means "is less than or more than" — in other words, G is *not* equal to X.)

```
20 LET X=INT (RND*6)
30 PRINT "THE COMPUTER HAS CHOSEN A
   NUMBER BETWEEN 0 AND 5. CAN YOU
   GUESS IT?"
40 INPUT G
60 IF G=X THEN PRINT "WELL DONE"
80 IF G <> X THEN PRINT
   "TOUGH LUCK — YOU'RE WRONG"
```

```
20 LET X=RND(6)-1
```

```
3Ø PRINT "THE COMPUTER HAS CHOSEN A
   NUMBER BETWEEN Ø AND 5. CAN YOU
   GUESS IT?"
4Ø INPUT G
6Ø IF G=X THEN PRINT "WELL DONE" ELSE
   PRINT "TOUGH LUCK—YOU'RE WRONG"
```

```
2Ø LET X=RND(6)—1
3Ø PRINT "THE COMPUTER HAS CHOSEN A
   NUMBER BETWEEN Ø AND 5. CAN YOU
   GUESS IT?"
4Ø INPUT G
6Ø IF G=X THEN PRINT "WELL DONE" ELSE
   PRINT "TOUGH LUCK – YOU'RE WRONG"
```

Microtip

Finding your way out

Beginners sometimes find it difficult to interrupt a RUNning program and get back to the program listing — perhaps because it needs amendment.

This usually happens when the computer is expecting a series of INPUTs. Whatever you type, it seems, simply fills the screen with rubbish.

First press CAPS SHIFT and SPACE to BREAK. If this doesn't work you could be in INPUT mode. If this is the case, use the cursor controls and DELETE to remove any quotation marks. Then use STOP (shifted A), then ENTER. This gives you report H — "stop in INPUT." Now press ENTER again to list the program.

Try pressing RUN/STOP, then type LIST. If this does not work, hold down RUN/STOP while you press RESTORE. Then type LIST.

Push ESCAPE, then type LIST. If this does not work, press the BREAK key and then type OLD, then LIST.

Press BREAK, then type LIST. If this does not work, push the RESET button located on the left side of the machine. Then type LIST.

```
2Ø LET X=INT(RND(1)*6)
3Ø PRINT "THE COMPUTER HAS CHOSEN A
   NUMBER BETWEEN Ø AND 5. CAN YOU
   GUESS IT?"
4Ø INPUT G
6Ø IF G=X THEN PRINT "WELL DONE"
8Ø IF G<>X THEN PRINT "TOUGH LUCK
   —YOU'RE WRONG"
```

RUN this program, and you will see that it is playable—but only just. To begin with, the screen is a bit cluttered. On top of that, the game expires after just one go.

To deal with the first problem, all you need do is enter:

```
1Ø CLS
5Ø CLS
```

```
1Ø PRINT "□"
5Ø PRINT "□"
```

This statement means "clear screen" so that only the new, relevant information appears there.

To deal with the second problem is slightly harder. One way round it would be to enter:

```
9Ø GOTO 1Ø
```

To re-start the game automatically.

STRING VARIABLES

A better way is to offer the player the chance of another game if he wants it. This may sound complicated, but in practice it is quite straightforward.

Start by typing in the complete program. On the ZX81, leave out OR A$="y" in Line 11Ø.

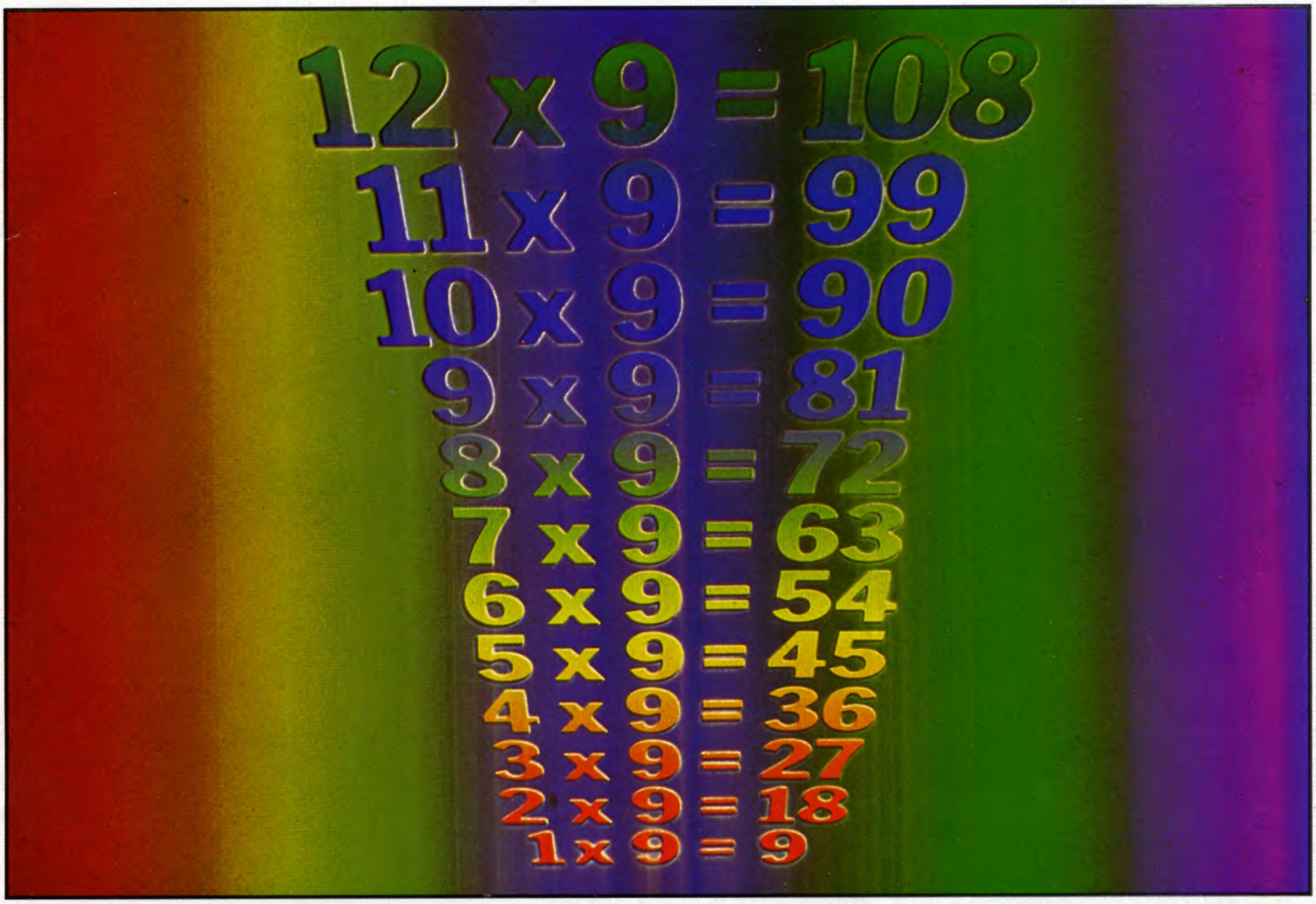
```
1Ø CLS
2Ø LET X=INT (RND*6)
3Ø PRINT "THE COMPUTER HAS CHOSEN A
   NUMBER BETWEEN Ø AND 5. CAN YOU
   GUESS IT?"
4Ø INPUT G
5Ø CLS
6Ø IF G=X THEN PRINT "WELL DONE"
7Ø IF G=X THEN GOTO 9Ø
8Ø PRINT "TOUGH LUCK—YOU'RE WRONG"
9Ø PRINT "DO YOU WANT ANOTHER GO? IF SO,
   PLEASE TYPE Y AND PRESS THE ENTER
   KEY"
1ØØ INPUT A$
11Ø IF A$="Y" OR A$="y" THEN GOTO 1Ø
12Ø GOTO 1ØØ
```

```
1Ø CLS
2Ø LET X=RND(6)—1
3Ø PRINT "THE COMPUTER HAS CHOSEN A
   NUMBER BETWEEN Ø AND 5. CAN YOU
   GUESS IT?"
4Ø INPUT G
5Ø CLS
6Ø IF G=X THEN PRINT "WELL DONE" ELSE
   PRINT "TOUGH LUCK—YOU'RE WRONG"
9Ø PRINT "DO YOU WANT ANOTHER GO? IF SO,
   PLEASE TYPE Y AND PRESS THE ENTER
   KEY"
1ØØ INPUT A$
11Ø IF A$="Y" THEN GOTO 1Ø
12Ø GOTO 1ØØ
```

```
1Ø CLS
2Ø LET X=RND(6)—1
3Ø PRINT "THE COMPUTER HAS CHOSEN A
   NUMBER BETWEEN Ø AND 5. CAN YOU
   GUESS IT?"
4Ø INPUT G
5Ø CLS
6Ø IF G=X THEN PRINT "WELL DONE" ELSE
   PRINT "TOUGH LUCK—YOU'RE WRONG"
9Ø PRINT "DO YOU WANT ANOTHER GO? IF SO,
   PLEASE TYPE Y AND PRESS THE RETURN
   KEY"
1ØØ INPUT A$
```



2. What punctuation means in programming. The lines at the top of the screen are the programmer's instructions; the other lines show how these have been carried out. Thus a comma (,) means 'tabulate'; a semi-colon (;) means 'close up tight'; and a single quotation mark (') on the Commodore, Spectrum and Acorn machines means 'begin a new line'.




```
110 IF A$="Y" THEN GOTO 10
120 GOTO 100
```

```
10 PRINT "♥"
20 LET X=INT (RND(1)*6)
30 PRINT "THE COMPUTER HAS CHOSEN A
   NUMBER BETWEEN 0 AND 5. CAN YOU
   GUESS IT?"
40 INPUT G
50 PRINT "♥"
60 IF G=X THEN PRINT "WELL DONE.":GOTO
   90
80 PRINT "TOUGH LUCK—YOU'RE WRONG."
90 PRINT "DO YOU WANT ANOTHER GO? IF SO,
   PLEASE TYPE Y AND PRESS THE RETURN
   KEY"
100 INPUT A$
110 IF A$= "Y" THEN GOTO 10
120 GOTO 100
```

As you can see, first you ask the player (in Line 90) if he wants another game. Then, to warn the computer to expect an answer, you use in Line 100 the INPUT statement.

This time, however, there is an important difference. After Line 20, the player entered a number. This time he is going to enter Y (for "yes") or N (for "no") — not a number, but a letter.

This means that at Line 100, instead of INPUT A, you must use INPUT A$.

The dollar sign is called a *string*, and A$ is known as a *string variable*.

Why is the $ necessary? To understand this, you need to know a great deal about how the computer stores and handles input — the subject of a later chapter.

For now, the important point to remember is:

When the computer is expecting a *number*, you use INPUT A, INPUT B, INPUT X or whatever.

When it is expecting a *letter* or *word*, you must use INPUT A$, INPUT B$, INPUT X$ or whatever.

Line 120 is included so that, if the player does not want another game immediately, the computer waits until he does by repeating the process until the answer does equal Y. It does this by repeatedly jumping back to Line 100 until a Y keypress after that line breaks the cycle.

KNOW YOUR TABLES?

The RND function has hundreds of uses in programming. Suppose, for example, you wanted a program to teach your eight-year-old son or brother his 9-times table. You could set it up like this:

```
10 PRINT "WHAT IS 1 TIMES 9?"
20 INPUT A
30 IF A=9 THEN PRINT "CORRECT"
40 PRINT "WHAT IS 2 TIMES 9?"
50 INPUT B
60 IF B=18 THEN PRINT "CORRECT"
```

...and so on.

But this would give you a very long program — without even solving the problem of how the computer responds if one of his answers is wrong!

Use the RND function, however, and you can produce a much more compact program that will not only ask all the right questions, but ask them in random order—

a much better way of teaching, as well as of programming.

Particularly when you are learning programming, it is always best to work out the 'core' of a program before adding the frills. So first try these lines (remember **NEW**!):

```
1Ø LET N=INT (RND*12+1)
2Ø PRINT "WHAT IS□"; N; "□TIMES 9?"
3Ø INPUT A
4Ø IF A=N*9 THEN PRINT "CORRECT"
```

```
1Ø LET N=RND(12)
2Ø PRINT "WHAT IS □"; N; "□ TIMES 9?"
3Ø INPUT A
4Ø IF A=N*9 THEN PRINT "CORRECT"
```

```
1Ø N = INT (RND(1)*12) + 1
2Ø PRINT "WHAT IS□"; N;"□TIMES 9?"
3Ø INPUT A
4Ø IF A=N*9 THEN PRINT "CORRECT"
```

This program is using **RND** in much the same way as in the guessing game. In Line 1Ø, you first set up a variable, or label, for the random number the computer selects. In this case it is **N**, but it could be any other letter or letters.



3. Computer symbols for basic arithmetic. The * sign, and not small x, is used for 'multiply.' The / means 'divided by,' and ↑ 'raised to the power of,' (The ZX81 uses ** instead). Plus and minus signs are conventional.

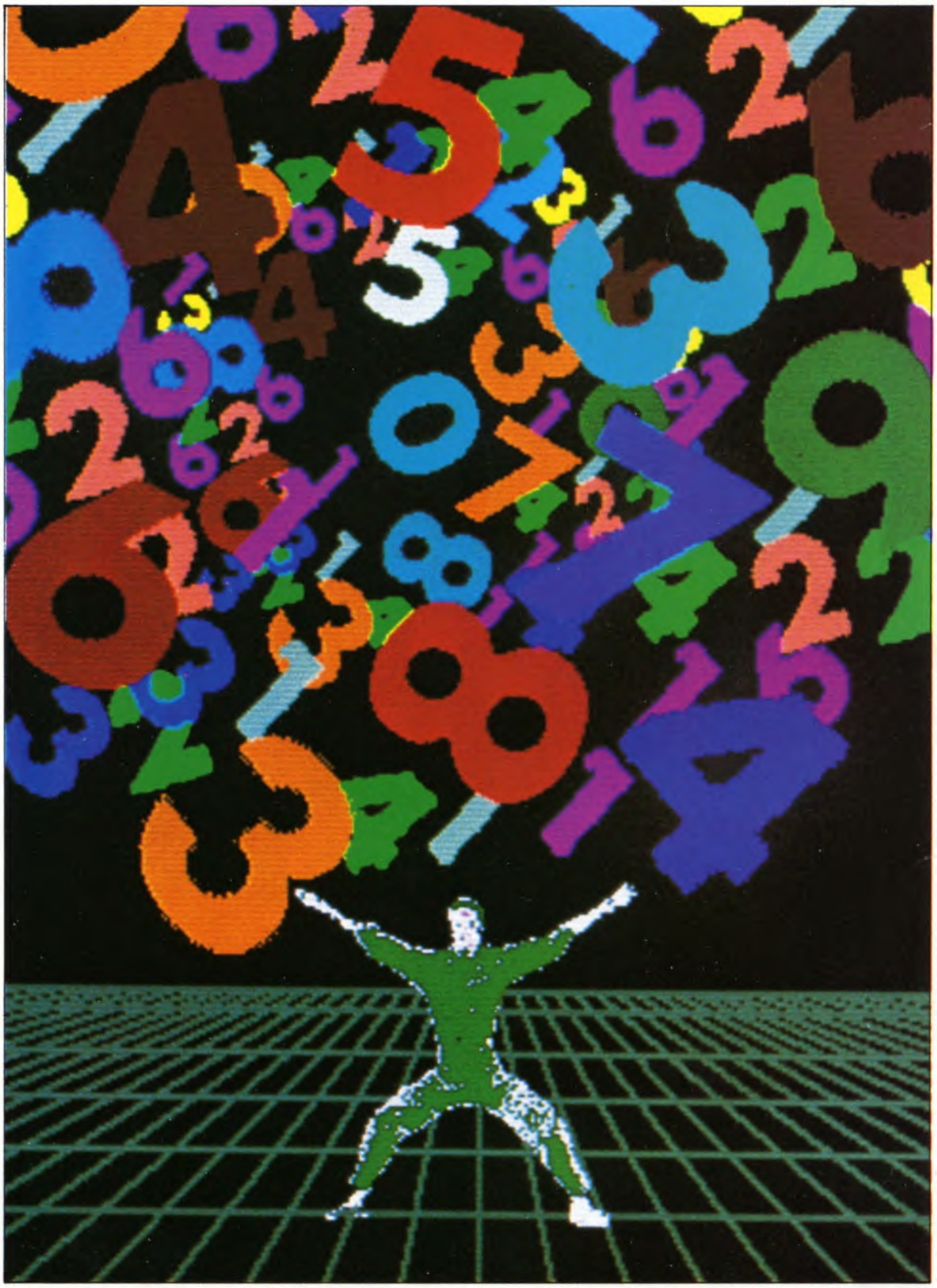
Then, in the right-hand half of Line 1Ø, you tell the computer to pick a number — any *whole* number — between 1 and 12. (The +1 on the Spectrum, ZX81 and Commodore is necessary because their random numbers start at Ø — a number not wanted for this job.)

In Line 2Ø you ask the player to multiply by 9 whichever number the computer has chosen this time. Line 4Ø warns the computer to multiply the random number by 9, compare this with the player's answer and — if the latter is correct — print "CORRECT".

RUN this program and you will find that it works — once. You could make it continuous by adding:

```
5Ø GOTO 1Ø
```

... but why not do the job properly, as here:



```
1Ø PRINT "HELLO. WHAT IS YOUR NAME?"
2Ø INPUT A$
3Ø CLS
4Ø PRINT "HELLO,□";A$, "I HAVE SOME","
   QUESTIONS FOR YOU"
5Ø PAUSE 2ØØ
6Ø CLS
7Ø LET N=INT (RND*12)+1
8Ø PRINT "WHAT IS□";N; "□TIMES 9?"
9Ø INPUT A
1ØØ IF A=N*9 THEN GOTO 15Ø
11Ø CLS
12Ø PRINT A;"□?"
13Ø PRINT "SORRY, PLEASE TRY AGAIN"
14Ø GOTO 8Ø
15Ø PRINT "WELL DONE,□"; A$,"HERE IS THE
   NEXT ONE"
16Ø PAUSE 15Ø
17Ø GOTO 6Ø
```

```
1Ø PRINT "HELLO, WHAT IS YOUR NAME?"
2Ø INPUT A$
3Ø CLS
4Ø PRINT "HELLO,□";A$:PRINT "I HAVE
   SOME QUESTIONS FOR YOU"
5Ø FOR X=1 TO 6ØØØ:NEXT X
6Ø CLS
7Ø LET N=RND(12)
8Ø PRINT "WHAT IS□";N;"□TIMES 9?"
9Ø INPUT A
1ØØ IF A=N*9 THEN GOTO 15Ø
11Ø CLS
12Ø PRINT A;"?"
13Ø PRINT "SORRY, PLEASE TRY AGAIN"
14Ø GOTO 8Ø
15Ø PRINT "WELL DONE,□";A$:PRINT "HERE
   IS THE NEXT ONE"
16Ø FOR X=1 TO 4ØØØ:NEXT X
17Ø GOTO 6Ø
```

```
1Ø PRINT "HELLO, WHAT IS YOUR NAME?"
2Ø INPUT A$
3Ø PRINT "♥"
4Ø PRINT "HELLO,□";A$:PRINT "I HAVE
   SOME QUESTIONS FOR YOU"
5Ø FOR X=1 TO 2ØØØ:NEXT X
6Ø PRINT "♥"
7Ø N=INT(RND(1)*12)+1
8Ø PRINT "WHAT IS□";N;"□TIMES 9?"
9Ø A=Ø: INPUT A
1ØØ IF A=N*9 THEN GOTO 15Ø
11Ø PRINT "♥"
12Ø PRINT A;"□?"
13Ø PRINT "SORRY, PLEASE TRY AGAIN"
14Ø GOTO 8Ø
15Ø PRINT "WELL DONE,□";A$:PRINT "HERE
   IS THE NEXT ONE"
16Ø FOR X=1 TO 2ØØØ:NEXT X
17Ø GOTO 6Ø
```

What most of these lines are doing will be obvious when you **RUN** the program. But a few are worth a comment:

Lines 3Ø, 6Ø and 11Ø are to stop the screen getting cluttered up with computer chatter or columns of wrong answers. (**RUN** the program with these lines deleted and you will see the difference they make.)



4. The numbers at the start of program lines are important. Without them, the computer will execute each line separately, rather than carrying out the program as a whole. However you enter them, the computer will automatically rearrange the lines in numerical order. Gaps between the numbers allow you to amend the program later

Lines 5Ø and 16Ø simply waste time, by making the computer 'count up to' a certain number before springing the next question on your unsuspecting infant. (How it does this is in BASIC Programming 2.)

In the Sinclair program the commas in Lines 4Ø and 15Ø, as all but absolute beginners will know, are to space out the messages so there are no word splits at the end of the line. In the programs for the other machines, the colons and extra PRINT statements in Lines 4Ø and 15Ø are doing the same job

From the viewpoint of your eight-year-old, this program has one terrible disadvantage: it goes on and on without stopping. You can release him from his misery, like this:

Push STOP, then ENTER

Push BREAK

Push ESCAPE

Push RUN/STOP

From your viewpoint, on the other hand, the program has a bonus feature:

Simply by changing the 9s in the program to 5s, 6s, 7s or whatever, you can test him on all his tables.

And because the computer does all the sums, you do not need to know *any* of the answers yourself!

Q+A

How do you specify a range of random numbers?

RND on the Sinclair, RND(1) on the Acorn, and RND(Ø) on the Dragon and Commodore generate a random number between Ø and Ø.999999. If you want a larger range of random numbers you have to multiply the original RND function. For example to generate a number between Ø and 39.999999, you would multiply the original function by 4Ø.

To generate a random *whole* number, as opposed to a *decimal* number, you use the integer or INT function. The expression now looks like this: INT (original function *4Ø). It will generate a number between Ø and 39 because the INT function merely cuts off all the figures after the decimal point.

Suppose you want to have a range of numbers from 1 to 4Ø. All you have to do is to add 1 to the INT expression. The Dragon and Acorn machines have a shortcut. A whole number between 1 and 4Ø is generated when RND(4Ø) is typed. See the table for some examples.

Specifying random number ranges

Generates a random number between Ø and Ø.999999	RND(1)	RND(1)	RND(Ø)	RND
Generates a random number between Ø and n*Ø.999999	RND(1)*n	RND(1)*n	RND(Ø)*n	RND*n
Generates a random number between −1Ø and +1Ø	RND(21)−11	INT(RND(1)*21)−1Ø	RND(21)−11	INT(RND*21)−1Ø
Generates a random whole number between Ø and 39	INT(RND(1)*4Ø)	INT(RND(1)*4Ø)	INT(RND(Ø)*4Ø)	INT(RND*4Ø)
Generates a random whole number between 1 and 4Ø	RND(4Ø)	INT(RND(1)*4Ø)+1	RND(4Ø)	INT(RND*4Ø)+1

SPEED UP YOUR GAMES ROUTINES

Machine code is not just for the experienced programmer. You can start off by using short routines to speed up your BASIC games—and have fun while you're learning

Machine code programming — on the surface, anyway — is far from easy. To most home computer owners, machine code is a daunting maze of numbers.

The best way to begin is by actually trying out some machine code routines. In this way you, right from the start, will be familiar with the advantages of machine code, and what it can do for your programs. Then, in later articles in the series, we will take you through all the mysterious numbers so that you can develop machine code programs of your own.

The graphics in this article use BASIC programs to put machine code routines into your computer's memory. In this way, you can produce far faster and more lifelike movement than you could using BASIC alone. Note that because this is dependent on the characteristics of individual computer's, the Sinclair and Commodore programs are designed *only* for the Spectrum and Commodore 64. Future articles will show how you can apply these techniques to the ZX81 and Vic 20.

Microtip

Watch the numbers!

The numbers which make up machine code routines must be typed into your computer with great accuracy.

Once a routine is in memory, you cannot normally recall it for amendment by using BASIC.

In this article, for example, the BASIC programs which make the tank and frog move can be changed at will. You can experiment, too, by altering the DATA which defines these two graphics, because by doing this you are only 're-shaping' the content of the pictures.

But the program which sets up the initial frames must be treated with great care. RUN it only when you have checked on the screen that your typing is 'spot on'.

To set up the graphics illustrated in figs 2 and 3 you will need to do three things.

First, you must set up in the computer's memory a frame, or grid, which defines the size of the graphic you want. At first, this frame will be reproduced as a series of *user defined graphics* characters (see below).

Second, you must enter a program which will allow you to move the frame around the screen.

Third, you must remove the user defined graphics characters in the frame and replace them with the graphic you actually want — tank, frog or whatever.

SPECTRUM UDG CHARACTERS

On the Spectrum a user definable graphics character, or UDG character for short, is a letter (such as A) which you can 'mould' into something else.



fig. 1

Each letter consists of 64 dots (some one colour, some another) within an 8 x 8 grid as in fig. 1. Provided you stay within the grid, you can write a program which changes each letter into any shape you like.

The Spectrum gives you 21 such letters: A to U inclusive. As fig. 2 shows, you can put these tiny grids together on the screen to form larger frames. At any one time, for example, you can have two 3 x 3 frames (with three UDG characters left over), or five 2 x 2 frames.

SETTING UP THE GRID

Both the frog and the tank illustrated in this article need a 3 x 3 frame of UDG characters, as under:

```
A B C
D E F
G H I
```

One way to create this would be to use a series of PRINT AT statements in BASIC:

```
PRINT AT 1Ø,1Ø;<graphics ABC>
PRINT AT 11, 1Ø;<graphics DEF>
```

A better way is to use the machine code routine which the BASIC program below sets up. When you type it in, make sure you use the Spectrum key words and not the actual letters. CODE, for example, is entered by pressing CAPS SHIFT and SYMBOL SHIFT together and then pressing the I key.

```
1Ø IF PEEK 23733=127 THEN CLEAR 32399:
   LET B=324ØØ:LET Z=Ø
2Ø IF PEEK 23733=255 THEN CLEAR 65199:
   LET B=652ØØ: LET Z=1
3Ø LET T=Ø:FOR N=B TO B+129:READ
   A:LET T=T+A:POKE N,A:NEXT N:READ
   A:IF A< >T THEN PRINT "DATA
   ERROR":STOP
4Ø IF Z=1 THEN POKE 65258,178: POKE
   65259,254: POKE 65277,179: POKE
   65278,254
5Ø SAVE "FramePrint" CODE B,13Ø
6Ø STOP
1ØØ DATA 24,55,1,22,Ø,Ø,32,32,32,22,Ø,Ø,
   32,32,32,22,Ø,Ø,32,32
11Ø DATA 32,22,Ø,Ø,144,145,146,22,Ø,Ø,147,
   148,149,22,Ø,Ø,15Ø,151,152,22
12Ø DATA Ø,Ø,153,154,155,22,Ø,Ø,156,157,
   158,22,Ø,Ø,159,16Ø,161,58,146,126
13Ø DATA 254,1,1,18,Ø,4Ø,8,56,4,2Ø3,33,
   24,2,14,Ø,221,33,147,126,221
14Ø DATA 9,58,137,92,71,62,24,144,221,
   119,1,6Ø,221,119,7,6Ø,221,119,13,58
15Ø DATA 136,92,71,62,33,144,221,119,2,
   221, 119,8,221,119,14,221,229,62,2,2Ø5
16Ø DATA 1,22,2Ø9,1,18,Ø,2Ø5,6Ø,32,2Ø1,9913
```

This routine, you may feel, is a very long-winded way of entering the equivalent of three PRINT AT statements. True, but it has two big advantages:

1 Once you have typed it once, you can SAVE it and use it over and over again.

2 In use, this routine will make the whole frame (or the graphic you replace it with) move about the screen much, much faster than the equivalent BASIC.

Without knowing machine code you cannot, of course, understand each of the



- SETTING UP A MOVEABLE GRAPHICS 'FRAME'
- GETTING FASTER MOVEMENT
- A FROG THAT JUMPS—AND A TANK THAT FIRES

above DATA numbers. But in summary:

Lines 1Ø and 2Ø work out where the program will be stored. (If you have a 16K machine the program is stored in a different part of the machine from where it would be in a 48K machine). Note that on some 48K machines which have been upgraded from 16K, this may not work. If you have a problem with a 48K machine, omit Line 1Ø and substitute Line 2Ø with:

```
2Ø CLEAR 65199: LET B=652ØØ:LET Z=1
```

Line 3Ø tells the computer to memorize the numbers out of DATA statements. If you make a mistake in copying this long list of numbers it will tell you there is a data error, so recheck your entries in Lines 1ØØ to 16Ø. These lines contain the numbers which, when placed in memory, form your machine code routine.

SAVEing this program is done in a different way from usual. When you have typed it and RUN it the screen will go blank and the computer will prompt you to start your tape recorder and then press "any key" to save the routine on tape. To reLOAD it, use LOAD "Frame Print" CODE.

MOVING THE GRID

With the program still in memory (or reLOADED from tape and RUN), you can begin to use it. First type NEW and ENTER so your new program does not corrupt the old. Then type in:

```
2Ø LET print=324ØØ: LET B=324Ø2: IF PEEK
   23733=255 THEN LET print=652ØØ: LET
   B=652Ø2
9Ø BORDER Ø: PAPER Ø: INK 4: CLS
1ØØ LET Y=8: LET X=15: LET Y1=8: LET
   X1=15: LET Z=1
11Ø LET A$=INKEY$
12Ø IF A$="z" AND X>Ø THEN LET
   X1=X-1: LET Z=1
13Ø IF A$="x" AND X<29THEN LET
   X1=X+1: LET Z=2
14Ø IF A$="p" AND Y>Ø THEN LET
   Y1=Y-1
15Ø IF A$="l" AND Y<18 THEN LET Y1=Y+1
17Ø LET X=X1: LET Y=Y1
18Ø PRINT AT Y,X;: POKE B,Z: RANDOMIZE
   USR print
19Ø GOTO 11Ø
```

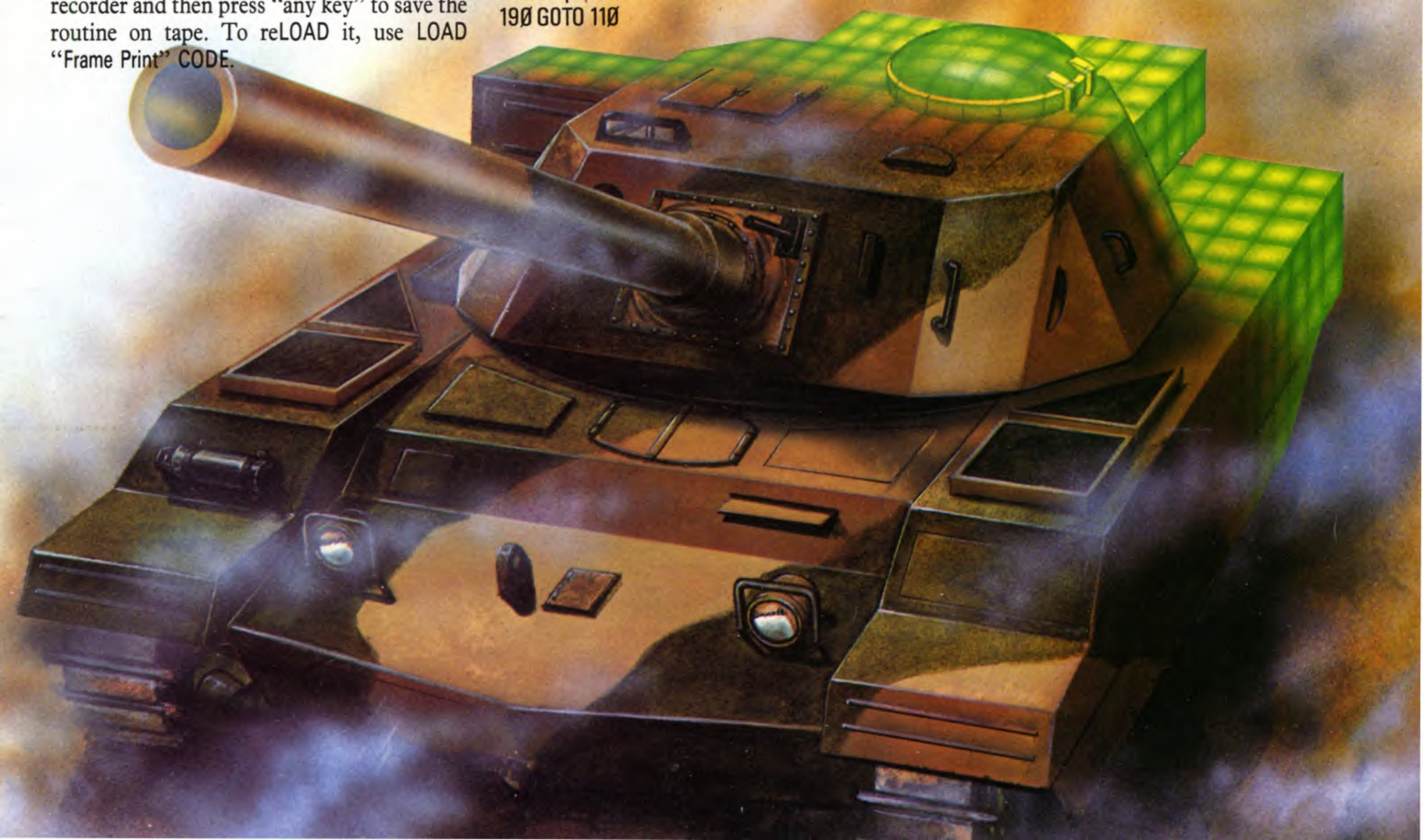
Note that print in lower case letters has to be typed in letter-by-letter, and is not the same as PRINT on the P key.

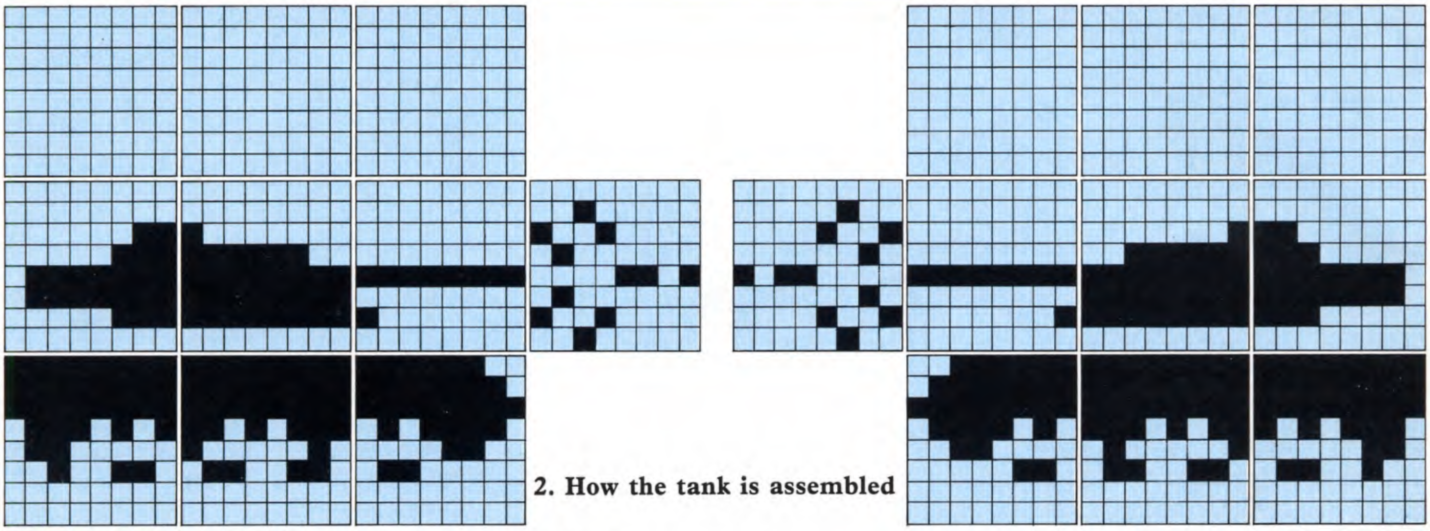
This BASIC program lets you move the frame around the screen using P-up; Z-left; L-down; X-right. When you RUN the program you will see that you have created not one 3 × 3 frame, but two, the second one consisting of the letters JKLMNOPQR. This is to allow you to have two versions of your final graphic—one facing left as it moves left, and one facing right as it moves right.

You will also notice something else: as the frames move, they leave behind a trail of unwanted characters. You do not need to use up any of your UDG characters to get rid of this clutter. All you need is:

```
16Ø PRINT AT Y,X;: POKE B, Ø: RANDOMIZE
   USR print
```

This produces a 3 x 3 frame of spaces which rub out whatever is on the screen at the Y,X position.





2. How the tank is assembled

CREATING THE TANK

To turn your existing frames into a tank, type in these few lines of statements plus DATA. They define a tank pointing to the left in frame 1, and to the right in frame 2.

```
10 FOR N=USR "a" TO USR "r"+7: READ A:
   POKE N,A: NEXT N
1000 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
1010 DATA 0,0,0,0,0,0,0,0,0,0,0,0,255,0,1,0
1020 DATA 0,0,1,63,255,255,255,0,0,0,
   192,224,254,254,224,0
1030 DATA 63,127,255,122,48,6,0,0,255,
   255,255,235,65,102,0,0
1040 DATA 255,255,255,174,6,100,0,0
1050 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
1060 DATA 0,0,0,0,0,0,0,0,0,0,3,7,127,127,
   7,0
1070 DATA 0,0,128,252,255,255,255,0,0,0,
   0,0,255,0,128,0
1080 DATA 255,255,255,117,96,38,0,0,255,
   255,255,215,130,102,0,0
1090 DATA 252,254,255,94,12,96,0,0
```

Two more UDG characters will 'arm' the tank. First press BREAK, then type:

```
10 FOR N=USR "a" TO USR "t"+7:READ A:
   POKE N,A: NEXT N
115 IF INKEYS="□" THEN GOTO 200
200 IF Z=2 THEN GOTO 300
210 FOR N=X−1 TO 0 STEP −1
220 PRINT INK 5;AT Y+1,N; CHR$ 162
230 PAUSE 1
240 PRINT AT Y+1,N;"□"
250 NEXT N
260 GO TO 110
300 FOR N=X+3 TO 31
310 PRINT INK 5;AT Y+1,N; CHRS 163
320 PAUSE 1
330 PRINT AT Y+1,N;"□"
340 NEXT N
350 GO TO 110
1100 DATA 0,4,9,2,176,2,9,4,0,32,144,64,
   13,64,144,32
```

RUN this program, and you can use the SPACE key to fire a shell.

CREATING THE FROG

To turn your existing frames into the frog, you will of course first have to get rid of the tank. So first type NEW, then ENTER.

What this does is to clear away both the tank and the program with which you controlled it. The machine code program which set up the frames, however, will still be in the Spectrum's memory — unless you have pulled the power plug.

If you *have* disconnected the power, first type CLEAR 32399 on the 16K machine, or CLEAR 65199 on the 48K machine. This reserves for the machine code a space at the top of the computer's memory, where the BASIC you are going to enter cannot reach — and corrupt — it. Next, type LOAD "" CODE and switch on your tape recorder to load the original machine code routine.

With the machine code loaded, type in the following statements plus DATA to define the frog. Then you can RUN the program:

```
30 RESTORE 5000: FOR N=USR "a" TO USR
   "r" +7: READ A: POKE N,A: NEXT N
5000 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
5010 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1
5020 DATA 0,0,0,0,0,128,192,176,0,0,0,0,0,
   0,0,0
5030 DATA 4,15,31,63,127,254.248,127.96.
   240,224,192,64,32,156,192
5040 DATA 0,0,0,0,0,0,0,0
5050 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,1,3,7
5060 DATA 8,28,27,70.255,254,252,249,0,
   0,0,0,0,0,0,0
5070 DATA 7,7,15,30,54,38,70,70,250,196,
   0,0,0,0,0,0
5080 DATA 0,0,1,1,3,6,2,0,140,144,16,32,
   32,48,32,0
5090 DATA 0,0,0,0,0,0,0,0
```

CONTROLLING THE FROG

To control the frog's movements, you need:

```
10 BORDER 0: PAPER 0: INK 4: BRIGHT 1: CLS
20 LET P=32400: IF PEEK 23733=255 THEN
   LET P=65200
100 PRINT AT 10,0;: RANDOMIZE USR P: IF
   INKEY$= "" THEN GOTO 100
110 RESTORE 1000: FOR F=1 TO 5
120 READ A,B,C: POKE P+2,A: PRINT AT B,C;:
   RANDOMIZE USR P
130 PAUSE 2: CLS : NEXT F
150 PRINT AT 10,12;: RANDOMIZE USR P: IF
   INKEY$="" THEN GOTO 150
200 FOR F=1 TO 5
210 READ A,B,C: POKE P+2,A: PRINT AT B,C;:
   RANDOMIZE USR P
220 PAUSE 2: CLS : NEXT F
230 GOTO 100
1000 DATA 1,10,0,2,7,3,2,5,6,2,7,9,1,10,12,
   1,10,12,2,7,15,2,5,18,2,7,21,1,10,24
```

To set up the graphics illustrated in figs 2 and 3 you will need to do three things.

First, you must set up in the computer's memory a frame, or grid, which defines the size of the graphic you want. To begin with, this frame will be reproduced as a series of *user defined graphics* characters (see below).

Second, you must enter a program which will allow you to move the frame around the screen. If your graphic is to be part of a game in BASIC, this movement routine will also normally be in BASIC.

Third, you must replace the user defined graphics characters in the frame with the graphic you actually want — tank, frog or whatever.

ACORN UDG CHARACTERS

On the Electron and BBC B, a user definable graphics character, or UDG for short, is a 'box' in which you can produce a picture.

Each 'box' contains 64 dots within an 8 x 8 grid. Since they are all the same colour as the background, the box is invisible to begin with. But provided you stay within the box, you can change the colours of some or all of the dots to form any shape you like.

The Acorn machines normally let you have 32 UDG characters. As fig 2 shows, you can put them next to each other on the screen to form larger shapes. At any one time, for example, you could have three 3 x 3 frames with five UDG characters left over. Or you could have eight 2 x 2 frames — or even one massive 8 x 4 frame with nothing left over.

SETTING UP THE GRID

Both the frog and the tank illustrated in this article need a 3 x 3 frame of UDG characters, as under:

224	225	226
227	228	229
23Ø	231	232

You could create this using a series of PRINT TAB commands, for example:

```
PRINT TAB (1Ø,1Ø); CHR$(224);CHR$(225);
   CHR$(226)
PRINT TAB (1Ø,11); CHR$(227);CHR$(228);
   CHR$(229) . . . and so on.
```

A better — if slightly longer — way is to use a BASIC program to set up a machine code routine, as shown below. This routine has two advantages: once you have entered it once, you can use it over and over again; and of course it allows you to move your graphics around the screen much faster than you could using BASIC alone.

Before entering the program, type NEW. Then type:

```
1Ø FOR T=&DØØ TO &D58
2Ø READ A:?T=A
3Ø NEXT T
4Ø *SAVE "FramePrint" ØDØØ +59
5Ø DATA 24,169,224,2Ø2,48,47,24Ø,7,
   2Ø2,24Ø,2,1Ø5,9,1Ø5,9,162,3,16Ø,3,32,
   238,255,24,1Ø5,1,136,2Ø8,247,2Ø2,
   24Ø,21,72,169
6Ø DATA 1Ø,32,238,255,169,8,32,238,255,
   32,238,255,32,238,255,1Ø4,76,17,
   &ØD,96,169,32,162,3,16Ø,3,32,238,
   255,136,2Ø8,25Ø
7Ø DATA 2Ø2,24Ø,24Ø,169,1Ø,32,238,
   255,169,8,32,238,255,32,238,255,32,
   238,255,169,32,76,57,&ØD
```

If you have a disk drive, this program will need some modification. Line 1Ø becomes....1Ø FOR T=&AØØ TO &A58. In Line 4Ø, replace....ØDØØ...with...ØAØØ. And in Lines 6Ø and 7Ø, replace...&ØD...with...&ØA.

Now RUN the program. When you have finished, the computer will tell you to press the 'record' button on your tape recorder and then press the computer's RETURN key. This will save the routine for immediate or future use.

DO NOT BREAK

Now a word of warning: do not use the BREAK key at any time while this machine code routine is in your computer's memory (as you might, for example, to amend a line of BASIC program or of DATA). Using the BREAK key will corrupt the routine and you may be unable to get your program back again.

Without knowing machine code you will not understand what the numbers in the DATA statements above mean, but you may find this helpful:

Lines 1Ø-3Ø take the numbers out of the DATA statement and place them into the memory of the machine.

Line 4Ø is the command needed to SAVE your machine code routine. You can change the filename if you wish.

MOVING THE GRID

Now you have a copy of the machine code routine on tape you can begin to use it.

First you should clear the memory of any 'rubbish,' so type NEW. (This does not get rid of the machine code routine, which is protected from the NEW statement, or any other BASIC, by its position in memory.)

Next, to move the frame around the screen, type in the following program:

```
1Ø MODE 1
2Ø VDU 23;82Ø2;Ø;Ø;Ø;
4Ø X=2Ø:Y=2Ø:X1=2Ø:Y1=2Ø:Z=1
5Ø A$=GET$
6Ø IF A$="Z" AND X>Ø THEN
   X1=X-1:Z=1
7Ø IF A$="X" AND X<37 THEN
   X1=X+1:Z=2
8Ø IF A$="L" AND Y<29 THEN Y1=Y+1
9Ø IF A$="P" AND Y>Ø THEN Y1=Y-1
12Ø X=X1:Y=Y1
13Ø VDU 31,X,Y:X%=Z:CALL &DØØ
14Ø GOTO 5Ø
```

(For disk drive, in Line 13Ø replace...&DØØ...with...&AØØ.)

You can now RUN the program and move the frame using P-up, Z-left, L-down and X-right.

When the program is RUN you still will not see very much. If you have just played a game that uses UDG characters, you may find that remnants of space ship or 'killer gorilla' have invaded your frames. Otherwise, all you are likely to see is a horizontal line when you move left. This is, of course, because you have not yet defined the UDG characters.

CREATING THE TANK

To make the program useful you can produce the tank in fig. 2 — pointing left in one frame and pointing right in another. Press ESCAPE, then enter:

```
3Ø GOSUB 26Ø
26Ø VDU 23,224,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,23,
   225,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø
27Ø VDU23,226,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,23,227,
   Ø,Ø,Ø,Ø,255,Ø,1,Ø
28Ø VDU 23,228,Ø,Ø,1,63,255,255,
   255,Ø,23,229,Ø,Ø,192,224,254,254,
   224,Ø
29Ø VDU 23,23Ø,63,127,255,122,48,
   6,Ø,Ø,23,231,255,255,255,235,65,
   1Ø2,Ø,Ø
```





```
300 VDU 23,232,255,255,255,174,6,
    100,Ø,Ø
31Ø VDU 23,233,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,23,234,
    Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø
32Ø VDU 23,235,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,23,236,
    Ø,Ø,3,7,127,127,7,Ø
33Ø VDU 23,237,Ø,Ø,128,252,255,
    255,255,Ø,23,238,Ø,Ø,Ø,Ø,255,Ø,128,Ø
34Ø VDU 23,239,255,255,255,117,
    96,38,Ø,Ø,23,24Ø,255,255,255,215,
    13Ø,1Ø2,Ø,Ø
35Ø VDU 23,241,252,254,255,94,12,
    96,Ø,Ø
38Ø RETURN
```

When you run this program you will notice that the tank leaves a trail behind it. Type this line in to get rid of it:

```
11Ø VDU 31,X,Y:X%=Ø:CALL &DØØ
```

(For disk drive, replace the . . . &DØØ . . . with . . . &AØØ.)

This line produces a 3x3 frame of space which rubs out what is on the screen at the X,Y position.

After seeing the tank you may want to create other graphics. This is quite easy, as long as you stay within the 3x3 frame.

In the meantime, you have used only 18 of the 32 available UDGs. You can make the tank fire a shell by defining two of the other UDGs as shells, one for firing left and the other for firing right. You need a few lines to control the shell. So *add* these lines to your existing program:

```
1ØØ IF A$="□" THEN GOTO 15Ø
15Ø IF Z=2 THEN GOTO 21Ø
16Ø FOR T=X−1 TO Ø STEP −1
17Ø VDU 31,T,Y+1,242,8
18Ø A$=INKEY$(5):VDU32
19Ø NEXT T
2ØØ GOTO 5Ø
21Ø FOR T= X+3 TO 39
22Ø VDU 31,T,Y+1,243,8
23Ø A$=INKEY$(5):VDU 32
24Ø NEXT T
25Ø GOTO 5Ø
36Ø VDU 23,242,Ø,4,9,2,176,2,9,4
37Ø VDU 23,243,Ø,32,144,64,13,64,144,32
```

As Line 1ØØ indicates, the space bar is your firing mechanism.

CREATING THE FROG

Creating the frog is a similar job to creating the tank. First you type NEW. This gets rid of the tank and its controlling program, but leaves the machine code routine intact. (This routine will go only when you turn off the computer or press [BREAK].)

If you have just turned on the computer, load up the routine from the tape you SAVEd earlier. Next, using a slight variation of the normal LOAD command, type *LOAD"" and the computer will load the program.

Now the computer is ready for the BASIC program needed to define the frog. So enter these lines:

```
3Ø VDU 23,224,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,23,225,Ø,
    Ø,Ø,Ø,Ø,Ø,Ø,Ø,23,226,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø
4Ø VDU 23,227,Ø,Ø,Ø,Ø,Ø,Ø,1,1,23,228,Ø,Ø,
    Ø,Ø, Ø,128,192,176
5Ø VDU 23,229,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,23,23Ø,4,
    15,31, 63,127,254,248,127
6Ø VDU 23,231,96,24Ø,224,192,64,32,156,
    192,23,232,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø
7Ø VDU 23,233,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,23,234,Ø,Ø,
    Ø, Ø,Ø,1,3,7
8Ø VDU 23,235,8,28,27,7Ø,255,254,252,
    249,23,236,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø
9Ø VDU 23,237,7,7,15,3Ø,62,54,7Ø,7Ø,23,
    238,25Ø,196,Ø,Ø,Ø,Ø,Ø,Ø
1ØØ VDU 23,239,Ø,Ø,1,1,3,6,2,Ø,23,24Ø,
    14Ø,144,16,32,32,48,32,Ø
11Ø VDU 23,241,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø
```

CONTROLLING THE FROG

These extra lines will make the frog jump when the space bar is pushed:

```
1Ø MODE 1
2Ø VDU 23; 82Ø2; Ø; Ø; Ø;
12Ø X=Ø:Y=2Ø:DY=Ø
13Ø VDU 31,X,Y
14Ø X%=1:CALL &DØØ
15Ø IF GET$=" "THEN GOSUB 18Ø
16Ø IF X>35 THEN VDU 31,X,Y:X%=Ø:CALL
    &DØØ:GOTO 12Ø
17Ø GOTO15Ø
18Ø RESTORE:FOR T=1 TO 4
19Ø VDU 31,X,Y−DY
2ØØ X%=Ø:CALL &DØØ
21Ø READ F,DY
22Ø X=X+3
23Ø VDU 31,X,Y−DY
24Ø X%=F:CALL &DØØ
25Ø A$=INKEY$(5)
26Ø NEXT T
27Ø RETURN
28Ø DATA 2,3,2,5,2,3,1,Ø
```

(For disk drive, change the . . . &DØØ . . . in Lines 14Ø, 16Ø, 2ØØ and 24Ø to . . . &AØØ.)

RUN the program. The DATA statements represent the height of the frog above 'ground' level.

To set up the graphics illustrated in figs 2 and 3 you will need to do three things:

First, you must set up in the computer's memory a frame which defines the size of the graphic you want. This frame is made up of *user defined graphics* characters which, on the Dragon, are invisible at first.

Second, you must enter the DATA which defines the shape of your tank or frog in the computer's memory (though not yet on the screen).

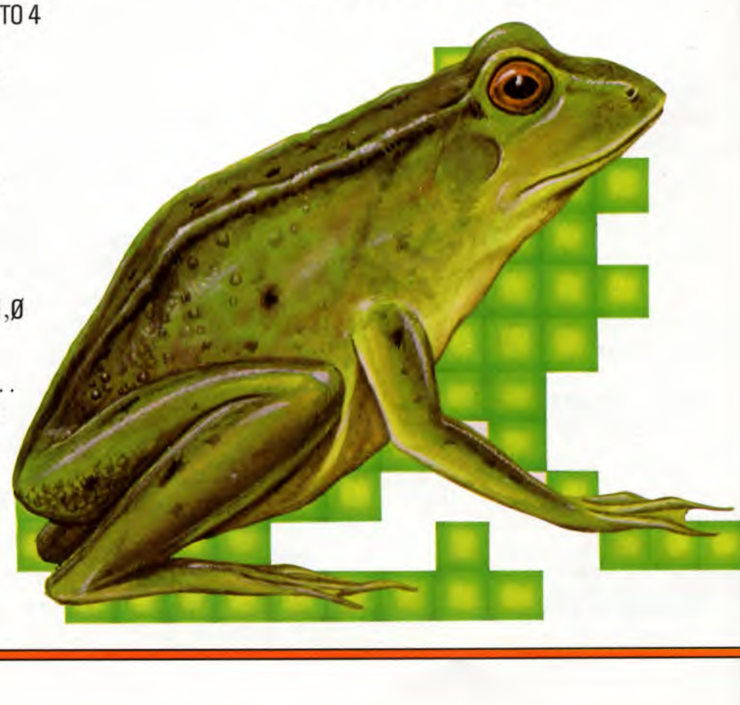
Third, you must enter another program which both prints the tank or frog and makes it move.

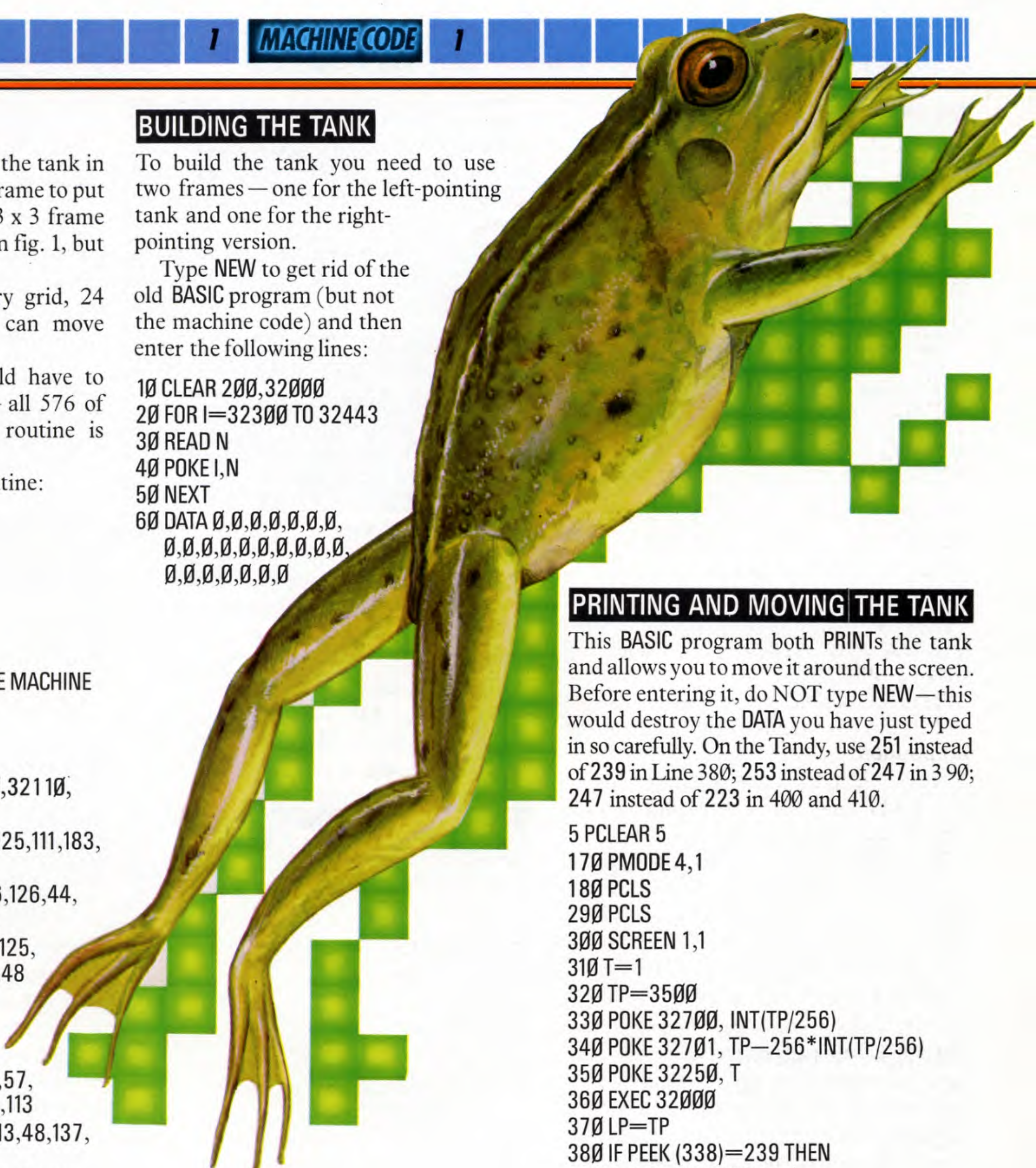
DRAGON UDG CHARACTERS

A user defined graphics character, or UDG for short, is a 'box' in which part of a picture can be located. Within this box there is an 8 x 8 grid of 64 dots, each of which, in PMODE 4,1, can be either black or white. (PMODE 4,1 is a much more flexible mode than the others, in which you have to define not just one pixel, or 'dot,' at a time, but two pixels or four.)

Putting several UDGs together allows you to build up much more detailed, larger shapes called *frames.* Unlike some other computers the Dragon has no inbuilt UDG characters, but parts of its memory will behave as UDGs if you write a program telling them to.

In theory, you have hundreds of these frames at your disposal. But using them all would involve hours and hours of programming and, in practice, you will usually use about five of them.





SETTING UP THE GRID

To produce the frog in fig. 3 or the tank in fig. 2 you first need to define a frame to put the character in. Both need a 3 x 3 frame of UDG characters—like that in fig. 1, but with nothing in it!

This gives you an imaginary grid, 24 dots by 24 dots, which you can move around the screen.

In Dragon BASIC, you would have to PSET every dot in the frame — all 576 of them. So the machine code routine is faster.

First type in the following routine:

```
10 CLEAR 200,32000
20 FOR I=32000 TO 32110
30 READ N
40 POKE I,N
50 NEXT
60 CLS
90 PRINT "PRESS ANY KEY TO SAVE MACHINE
   CODE ROUTINE"
100 B$=INKEY$
110 IF B$="" THEN 100
120 CSAVEM"FRAMEPRN",32000,32110,
   32000
130 DATA 190,127,188,134,3,183,125,111,183,
   125,112,134,8,183,125,113
140 DATA 182,125,250,39,50,206,126,44,
   74,198,72,61,51,203,166,192
150 DATA 167,132,48,136,32,122,125,
   113,38,244,134,8,183,125,113,48
160 DATA 137,255,1,122,125,111,
   38,230,134,3,183,125,111,48,
   137,0
170 DATA 253,122,125,112,38,216,57,
   95,231,132,48,136,32,122,125,113
180 DATA 38,246,134,8,183,125,113,48,137,
   255,1,122,125,111,38,232
190 DATA 134,3,183,125,111,48,137,0,253,
   122,125,112,38,218,57
```

Now you can SAVE the routine for immediate or future use. The computer will prompt you to do this, so have your tape recorder handy.

As well as producing faster graphics than the equivalent BASIC program could, this machine code routine is shorter and so takes up less memory space.

Until you understand 6809 machine code you cannot know what each number in the DATA means. The overall 'picture', however, is that—

Line 10 sets aside some memory for the routine, at the same time protecting it from corruption by any BASIC that you enter later.

Lines 90 to 120 SAVE the machine code from memory onto tape.

BUILDING THE TANK

To build the tank you need to use two frames—one for the left-pointing tank and one for the right-pointing version.

Type NEW to get rid of the old BASIC program (but not the machine code) and then enter the following lines:

```
10 CLEAR 200,32000
20 FOR I=32300 TO 32443
30 READ N
40 POKE I,N
50 NEXT
60 DATA 0,0,0,0,0,0,0,
   0,0,0,0,0,0,0,0,0,0,
   0,0,0,0,0,0,0
70 DATA 0,0,3,7,127,127,7,0,0,0,128,252,
   255,255,255,0
80 DATA 0,0,0,0,255,0,128,0,255,255,
   255,117,96,38,0,0
90 DATA 255,255,255,215,130,102,0,0,252,
   254,255,94,12,96,0,0
100 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
   0,0,0,0,0,0,0
110 DATA 0,0,0,0,255,0,1,0,0,0,1,63,255,
   255,255,0
120 DATA 0,0,192,224,254,254,224,0,63,
   127,255,122,48,6,0,0
130 DATA 255,255,255,235,65,102,0,0,
   255,255,255,174,6,100,0,0
```

This defines the tank in the computer's memory, but still does not print it on the screen. So the final stage is:

PRINTING AND MOVING THE TANK

This BASIC program both PRINTs the tank and allows you to move it around the screen. Before entering it, do NOT type NEW—this would destroy the DATA you have just typed in so carefully. On the Tandy, use 251 instead of 239 in Line 380; 253 instead of 247 in 390; 247 instead of 223 in 400 and 410.

```
5 PCLEAR 5
170 PMODE 4,1
180 PCLS
290 PCLS
300 SCREEN 1,1
310 T=1
320 TP=3500
330 POKE 32700, INT(TP/256)
340 POKE 32701, TP-256*INT(TP/256)
350 POKE 32250, T
360 EXEC 32000
370 LP=TP
380 IF PEEK (338)=239 THEN
   TP=TP-32:GOTO 440
390 IF PEEK (342)=247 THEN TP=TP+32:
   GOTO440
400 IF PEEK (340)=223 THEN
   TP=TP-1:T=2:GOTO 440
410 IF PEEK (338)=223 THEN TP=TP+1:
   T=1: GOTO 440
430 GOTO 380
440 IF TP<1536 OR TP>6941 THEN TP=LP
470 GOTO 330
```

Now you can RUN the program and move the tank around the screen using P-up, Z-left, L-down and X-right.

You will notice that the tank leaves a trail behind it. To get rid of it, type in these lines.

```
450 POKE 32250,0
460 EXEC 32000
```

This uses a special feature of the machine code routine that prints a frame of blanks.

FIRING A SHELL

To make things more interesting you can add a bit more BASIC program to move a shell fired from the tank. Enter these lines and RUN the program—the space bar will then fire the shell. On the Tandy you need to type in 247 instead of 223 in Line 42Ø, or the program will not work:

```
16Ø DIM A(2), B(2), C(2)
19Ø FOR I=1536 TO 176Ø STEP 32
2ØØ READ N
21Ø POKE I, N
22Ø NEXT
23Ø GET (Ø,Ø)—(7,7), A
24Ø FOR I=1536 TO 176Ø STEP 32
25Ø READ N
26Ø POKE I, N:NEXT
28Ø GET (Ø,Ø)—(7,7), B
42Ø IF PEEK (345)=223 GOSUB 5ØØ
48Ø DATA Ø,32,144,64,13,64,144,32
49Ø DATA Ø,4,9,2,176,2,9,4
5ØØ IF T=2 THEN 56Ø
51Ø YP=INT ((TP—1536)/32)+8
52Ø XP=8*(TP—1533—(YP—8)*32)
53Ø IF XP>255 THEN 62Ø
54Ø PUT (XP,YP)—(XP+7,YP+7), A
55Ø GOTO 6ØØ
56Ø YP=INT ((TP—1536)/32)+8
57Ø XP=8*(TP—1537—(YP—8)*32)
58Ø IF XP<Ø THEN 62Ø
59Ø PUT (XP,YP)—(XP+7,YP+7), B
6ØØ FOR I=1TO1ØØ:NEXT
61Ø PUT (XP,YP)—(XP+7,YP+7), C
62Ø RETURN
```

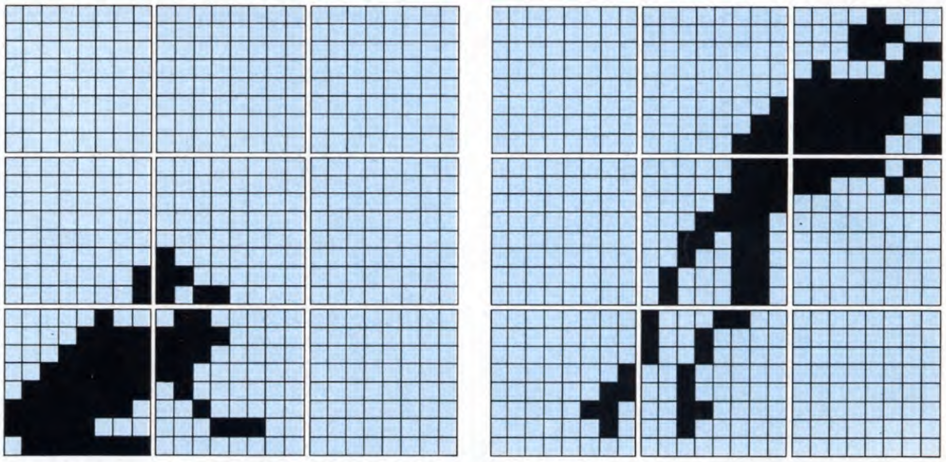
CREATING THE FROG

Fig. 3 shows the frames that you can define now and which, when controlled properly, will make the frog jump across the screen.

Unless you have just turned on the computer, the machine code routine should still be in memory. If it isn't allocate some memory for it by typing CLEAR2ØØ,32ØØØ. Then type CLOADM and turn on your tape recorder.

To define the frog you need these lines. First type NEW to get rid of any "rubbish" in the machine, and then enter this DATA:

```
1Ø CLEAR 2ØØ,32ØØØ
2Ø FOR I=323ØØ TO 32443
3Ø READ N
4Ø POKE I, N:NEXT
6Ø DATA Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,
   Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø
7Ø DATA Ø,Ø,Ø,Ø,Ø,Ø,1,1,Ø,Ø,Ø,Ø,Ø,128,
   192,176
8Ø DATA Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,4,15,31,63,127,
   254,248,127
9Ø DATA 96,24Ø,224,192,64,32,156,192,
   Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø
```



3. The frog graphics. The large squares are 8 pixels by 8

```
1ØØ DATA Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,1,3,
    7,8,28,27,7Ø,255,254,252,249
11Ø DATA Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,7,7,15,30,62,54,
    7Ø,7Ø
12Ø DATA 25Ø,196,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,1,1,3,
    6,2,Ø
13Ø DATA 14Ø,144,16,32,32,48,32,Ø,Ø,Ø,Ø,
    Ø,Ø,Ø,Ø,Ø
```

CONTROLLING THE FROG

To make the frog jump, type NEW, then enter the following. On the Tandy use 247 instead of 223 in Line 1ØØ:

```
1Ø PCLEAR 5
2Ø PMODE 4,1
3Ø PCLS
4Ø SCREEN 1,1
5Ø FP=4487
6Ø POKE 327ØØ,17
7Ø POKE 327Ø1,135
8Ø POKE 3225Ø,1
9Ø EXEC 32ØØØ
1ØØ IF PEEK(345)<>223 THEN 1ØØ
11Ø RESTORE
12Ø FOR I=1 TO 5
13Ø READ F,FD
14Ø POKE 327ØØ,INT (FP/256)
15Ø POKE 327Ø1,FP—256*INT (FP/256)
16Ø POKE 3225Ø,F
17Ø EXEC 32ØØØ
18Ø FOR J=Ø TO 7Ø:NEXT
19Ø POKE 3225Ø,Ø
2ØØ EXEC 32ØØØ
21Ø FP=FP+FD
22Ø NEXT
23Ø GOTO 8Ø
24Ø DATA 1,—287,2,—253,2,258,2,291,1,Ø
```

The DATA statement at the end of the program contains the numbers that make the frog jump. As you will see, the frog jumps in an arc across the screen, giving the effect of movement when you tap the space bar.

The best way to create graphics similar to those in figs 1 and 2 is to use one of the Commodore 64's special graphics features: *sprites*, a 'king-sized' type of UDG.

CREATING THE TANK

The sprite program here covers three activities: an initializing routine, a control sequence to regulate the sprite's movement on the screen, and data which is used to define the shape of the tank sprite:

```
1Ø FOR M=252 TO 253:FOR I=64*M TO
   64*M+63: POKE I,Ø: NEXT
2Ø FOR I=64*M+21 TO 64*M+56:READ
   A:POKE I,A: NEXT:NEXT
6Ø SC=53248:X3=24:X=24:Y3=157:
   Y=157
7Ø POKE 2Ø43,252:POKE 2Ø42,254:POKE
   SC+23,Ø:POKE SC+29,Ø:POKE
   SC+42,12:POKE SC+27,Ø
9Ø J=Ø:R=Ø:POKE 65Ø,128:PRINT"♥":
   POKE SC+21,8
1ØØ GET A$:IF A$=""THEN 18Ø
11Ø IF A$="Z"AND X3>24 THEN
    X3=X3—12
12Ø IF A$="X"AND X3<31Ø THEN
    X3=X3+12
13Ø IF A$="P" AND Y3>5Ø THEN
    Y3=Y3—12
14Ø IF A$="L"AND Y3<22Ø THEN
    Y3=Y3+12
18Ø IF X<X3 THEN X=X+3:POKE 2Ø43,253
19Ø IF X>X3 THEN X=X—3:POKE 2Ø43,252
2ØØ IF Y<Y3 THEN Y=Y+3
21Ø IF Y>Y3 THEN Y=Y—3
22Ø XA=INT(X/256):XB=X—XA*256
23Ø POKE SC+6,XB:POKE SC+7,Y
24Ø POKE SC+16,4*(1 AND JA)+8*
    (1 AND XA)
25Ø GOTO1ØØ
```

```
1ØØØ DATA Ø,1,192,Ø,63,224,255,255,254,
    Ø,255,254,1,255,224,Ø,Ø,Ø
1Ø1Ø DATA 63,255,255,127,255,255,255,
    255,255,122,235,174,48,65,6
1Ø2Ø DATA 6,1Ø2,1ØØ
1Ø3Ø DATA 3,128,Ø,7,252,Ø,127,255,255,
    127,255,Ø,7,255,128,Ø,Ø,Ø
1Ø4Ø DATA 255,255,252,255,255,254,
    255,255,255,117,215,94,96,13Ø,12
1Ø5Ø DATA 38,1Ø2,96
```

If you now RUN the program, you can see a tank sprite displayed on the screen. You can use the Z and X keys to move it left or right, and the P and L keys to move it up and down. Press RUN/STOP to terminate the program. Notice how the tank sprite changes shape when the direction of movement is altered.

ADDING SOME ACTION

So far you have used only one of the available sprites, and another sprite can be created to add some firepower to the tank. This shell fires across the screen when you tap the space bar, and is obtained by adding the following program lines:

```
3Ø FOR M=254 TO 255:FOR I=64*M TO
    64*M+62:POKE I,Ø:NEXT
4Ø M2=Ø:IF M=255 THEN M2=2
5Ø FOR I=64*M+18 TO 64*M+36
    STEP3:READA: POKE I+M2,A: NEXT: NEXT
8Ø POKE SC+23,Ø:POKE SC+29,Ø:POKE
    SC+41,1
15Ø IF A$<>"□"THEN 18Ø
16Ø RT=(PEEK(2Ø43)=253):X2=-36Ø*RT:
    Y2=Y:J=X-24*RT-11:POKE
    2Ø42,254-RT
17Ø POKE SC+21,12: GOSUB 26Ø
26Ø FOR T=J TO X2-4 STEP(RT*2+1)*-2Ø
27Ø JA=INT(T/256):JB=T-JA*256
28Ø POKE SC+16,4*(1 AND JA)+8*(1 AND
    XA)
29Ø POKE SC+4,JB:POKE SC+5,Y
3ØØ FOR P=1 TO 3Ø:NEXT: NEXT:RETURN
15ØØ DATA 4,9,2,176,2,9,4
151Ø DATA 32,144,64,13,64,144,32
```

To SAVE the program for future use, first type POKE 53269,Ø (which clears the screen of sprites) and then use a normal BASIC program SAVE.

If you try to SAVE without first clearing the sprites, they will corrupt your program.

CREATING A FROG SPRITE

Now let's try the frog. The graphics of the frog shown in fig. 2 have to be changed slightly so that each can be accommodated in a single sprite. This is done by removing three of the 24 rows, and the only effect of this is to shorten the legs a little.

Either switch off and then back on or simply type NEW RETURN and enter this:

```
1Ø FOR M=252 TO 253:FOR I=64*M TO
    64*M+29:POKE I,Ø:NEXT
2Ø FOR I=64*M+3Ø+3Ø*(M=253) TO
    64*M+62:READ A:POKE I,A:NEXT:NEXT
3Ø PRINT"♥"
4Ø SC=53248:X=24:Y=155:POKE 2Ø43,
    252:POKE SC+7,Y:POKE SC+6,X:POKE
    SC+16,Ø
5Ø POKE SC+23,Ø:POKE SC+29,Ø:POKE
    SC+27,Ø:POKE SC+42,5:POKE SC+21,8
6Ø SC=53248:X=24:Y=155:POKE
    2Ø43,252:POKE SC+7,Y:POKE
    SC+6,X:POKE SC+16,Ø:YR=Ø
7Ø GET A$
8Ø IF Y>155 THEN POKE 2Ø43,252
9Ø IF PEEK (2Ø43)=253 THEN 12Ø
1ØØ IF A$<>"□"THEN 7Ø
11Ø POKE 2Ø43,253:YR=7.5
12Ø Y=Y-YR:YR=YR-.7
13Ø X=X+4:IF X=296 THEN POKE
    2Ø43,252:FOR T=1 TO 2ØØ: NEXT:
    GOTO 6Ø
14Ø XA=INT(X/256):XB=X-XA*256
15Ø POKE SC+6,XB:POKE SC+7,Y:POKE
    SC+16,(1 AND XA)*8:GOTO7Ø
1000 DATA 0,128,0, 1,192.0, 1,176,0
1Ø1Ø DATA 4,96,Ø, 15,24Ø,Ø, 31,224,Ø,
    63,192,Ø
1Ø2Ø DATA 127,64,Ø, 254,32,Ø, 248,156,Ø,
    127,192,Ø
1Ø3Ø DATA Ø,Ø,28, Ø,Ø,27, Ø,Ø,7Ø, Ø,Ø,255,
    Ø,1,254, Ø,3,252, Ø,7,249
1040 DATA 0,7,250,0,7,196,0,15,0,0,30,0,
    0,62,0,0,54,0,0,70,0
1050 DATA 0,14,0,0, 0,144,0, 1,16,0, 1,16,0,
    1,32,0, 3,32,0, 6,48,0, 2,32,0
```

Tap the space bar to make the frog jump.



Q+A

What's a sprite?

A *sprite* is a kind of high resolution user defined graphic (UDG), also referred to as a *movable object block* (MOB). It is used as a kind of mobile component of high quality Commodore 64 graphics, in preference to UDGs. Unlike a UDG, a sprite offers smooth, easily programmable pixel-by-pixel movement in any direction, just as if it were a single character even though its size is 24 x 21 pixels (compared to 8 x 8 for a standard character).

Normally up to eight sprites can be displayed together anywhere on the screen but the number can be increased by special programming. A group of sprites can be arranged side by side, or overlaid for 3D effects. Sprites can be easily doubled in size, set for collision detection, adopt hi-res or multicolour forms — in short, they are extremely versatile.

Programming for sprites actually proves simpler than that for UDGs if sophisticated animation sequences are required. For example, you don't have to follow the trail of a sprite with a blanking out routine to 'erase' what's been printed in a previous position. And sprites may be used with any of the other graphics modes. Each one's characteristics can be changed at will.

THE ART OF THE FOR... NEXT LOOP

All that a FOR... NEXT loop does is to make your computer count up to a certain number, then stop. But you can use it in everything from games to business programming

The FOR . . . NEXT loop takes the hard work out of many repetitive operations. It is used when you want the computer to count for you, usually executing some other operations as it goes, then stop when it reaches a predetermined number.

You can create your own "instant paintings" using FOR . . . NEXT loops in quite tiny programs. You'll also find them useful in games programming—indeed, in programs of all kinds.

WHAT IS A FOR . . . NEXT LOOP

A FOR . . . NEXT loop in BASIC is a device which makes the computer repeat the same operation a number of times.

Suppose, for example, you wanted to know the square roots of all the numbers from 1 to 100. You could tell the computer:

```
PRINT SQR(1)
PRINT SQR(2)
PRINT SQR(3)
```

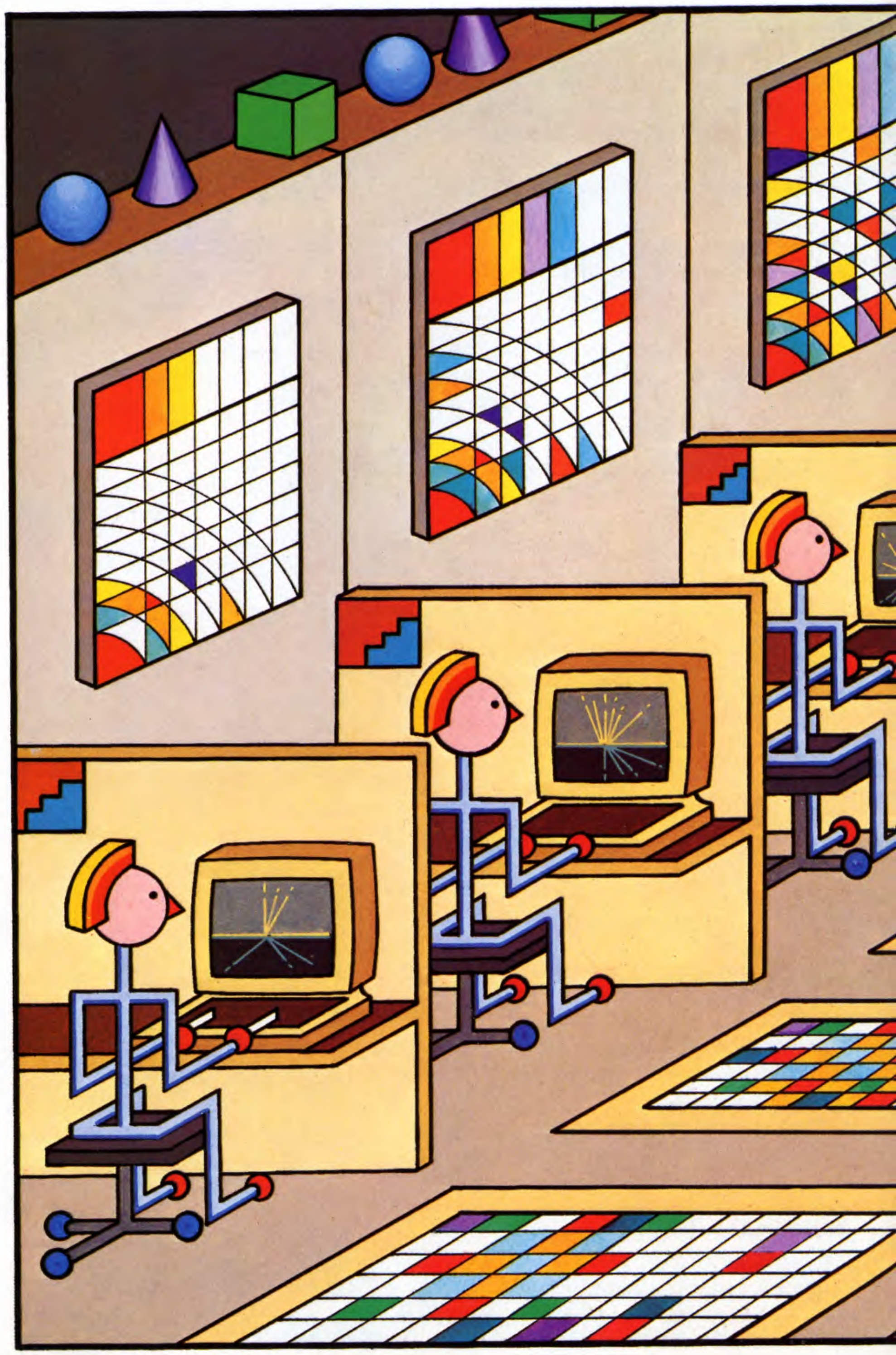
. . . and so on. And each time you asked a question, the computer would display the answer. Quite apart from imposing unnecessary wear and tear on your typing finger(s), this is not a particularly efficient use of the computer. So:

```
10 FOR n=1 TO 100
20 PRINT n, SQR n
30 NEXT n
40 PRINT "and that is all"
```

```
10 FOR N=1 TO 100
20 PRINT N;"□";SQR(N)
30 NEXT N
40 PRINT "AND THAT IS ALL"
```

```
10 FOR N=1 TO 100
20 PRINT N, SQR (N)
30 NEXT N
40 PRINT "AND THAT IS ALL"
```

What this tells the computer to do is to print 1 and its square root, 2 and its square root, 3 and its square root . . . and so on until it reaches 100, when it stops.



- 'COUNTING UP TO 100' BY COMPUTER
- THE **FOR** . . . **NEXT** LOOP AS A DELAYING DEVICE
- CREATING SOUND EFFECTS
- PAINTING BY NUMBERS WITH **FOR** . . . **NEXT** LOOPS
- SUNRISE PATTERNS
- INSTANT PATCHWORK AND EMBROIDERY



How does it do this? When the computer encounters FOR . . . it knows that the following lines of the program are going to be repeated. So it *executes* (carries out) all the following lines until it reaches . . . NEXT. Then it loops back to the line with FOR . . . in it and repeats the process line by line.

While the computer is doing this, it is also counting. The first time it reaches Line 2Ø it calculates the square root of 1, the second time around it calculates the square root of 2, and so on.

Once it has dealt with the highest number in the FOR . . . statement, the computer automatically quits the loop and goes on to the next line of the program—in this case, Line 4Ø.

FRACTIONS, TOO

In carrying out a FOR . . . NEXT instruction, the computer can count in units other than 1. Try this, for example:

```
1Ø FOR n=1 TO 3Ø STEP 2.7
2Ø PRINT n, SQR n
3Ø NEXT n
```

```
1Ø FOR N=1 TO 3Ø STEP 2.7
2Ø PRINT N;"□"; SQR(N)
3Ø NEXT N
```

```
1Ø FOR N=1 TO 3Ø STEP 2.7
2Ø PRINT N, SQR(N)
3Ø NEXT N
```

The computer, you'll notice, is not at all deterred by the fact that 3Ø will not divide evenly in the STEPs you have asked for. It just goes as near as it can, then stops.

Nor is it bothered by the number of lines between the FOR . . . part of the loop and the NEXT part. You can write for FOR . . . into Line 1Ø and the NEXT into Line 9Ø—or even 9ØØØ if you like—and your computer will faithfully remember it.

Remember, though, that it will execute *all* the lines within the loop every time it passes through.

DELAYING ACTION

FOR . . . NEXT loops have dozens of uses in programming. And the easiest of them is simply to waste time.

If you refer back to the 'Know your tables?' program on page 7 you will find a good example.

In this case, all that happens between each FOR . . . and its NEXT is that the computer 'counts up' one number. Computers count very rapidly, so this takes only a tiny fraction of a second (the exact time varies from computer to computer, but is in the order of hundredths or thousandths). But by the time it has counted to 1ØØØ you have a noticeable pause. And if you should program it:

```
1Ø FOR N=1 to 1ØØØØØØ
2Ø NEXT N
```

. . . you would probably have time for a coffee before it executed the next line in the program.

Q+A

How can I keep track of the variables in my programs, and not get lost among all those Xs and Ys?

Keeping track of your variables is much easier if you give them names that actually mean something in English. In a short program, your variables are easy to trace. But in a longer program if, for example, you are setting out material on the screen, it is easier to remember FOR row...and FOR column...than, say, FOR x...and FOR y. Some computers recognise only one- or two-letter variables, discarding any other letters. In this case, try using suitable initials — T for time, SC for score, HS for high score, H for hits and so on. This makes them very easy to pick up.

PEEL APPEAL

Games programmers often make such pauses less boring by inserting a few notes of a tune into them. Try this, for example:

```
1Ø FOR n=29 TO 1Ø STEP −1
2Ø BEEP .Ø15, n
3Ø NEXT n
```

```
1Ø FOR N=16Ø TO 1ØØ STEP −4
2Ø SOUND 1,−15, N, 1
3Ø NEXT N
```

```
1Ø FOR N=12 TO 1 STEP−1
2Ø PLAY "T4Ø;Ø4;"+STR$ (N)
3Ø NEXT N
```

This is the noise that programmers sometimes use to say 'You've failed' or 'The alien has landed'. It illustrates one point to remember: when you are counting downwards from a given starting point, your **STEP**s must be *minus* 1 (or whatever)—not just 1.

(The Commodore 64 does not have a simple sound command. You would have to write a separate sound generating routine between Lines 1Ø and 3Ø, branched to by a **GOSUB** in Line 2Ø).

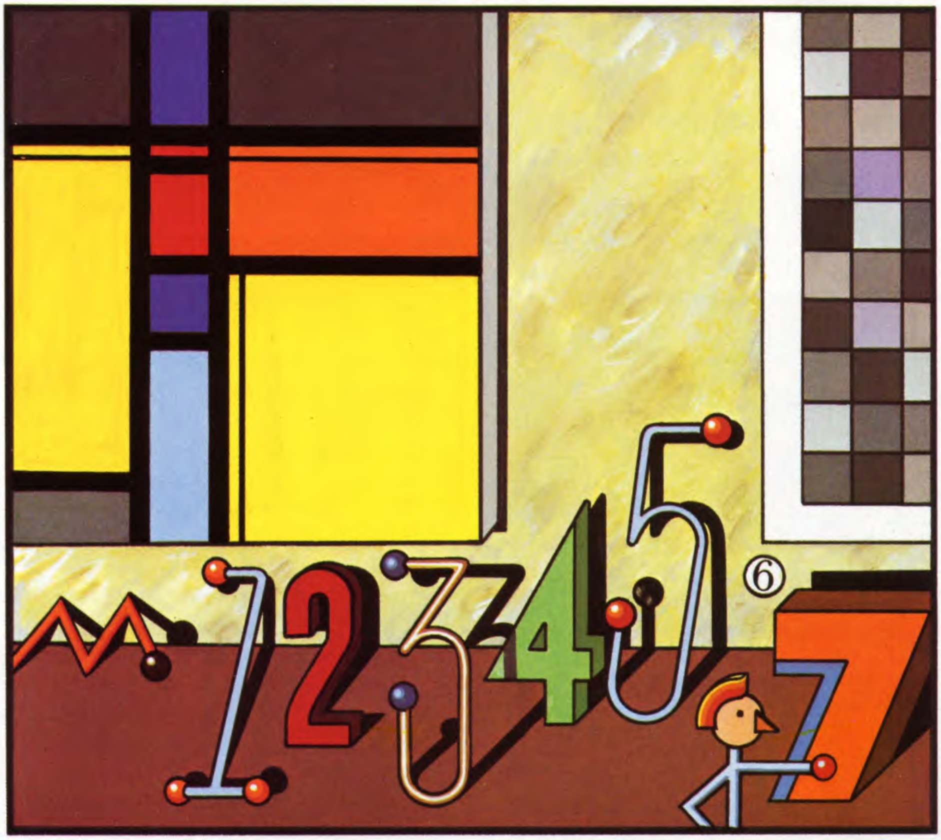
Microtip

Use the right loop

There are many occasions for using a **FOR...NEXT** loop—and just as many occasions when you shouldn't. The rule is:

When you want a program sequence performed a fixed number of times, without breaking into it at any stage, use a **FOR...NEXT** loop.

When you want a program sequence executed only until some condition is fulfilled, and then want to break out of the loop, use a different statement instead. Most often this will be **GOTO**, taking you back to an earlier line to complete a loop. But on some computers you can use **REPEAT...UNTIL** instead.



PAINTING BY NUMBERS

Just for fun—plus valuable practice in programming—you can use **FOR...NEXT** loops to create a huge range of graphics effects. Here is one example:

```
1Ø FOR n=Ø TO 21
2Ø LET m=RND*31
3Ø INK RND*7+1
4Ø PRINT AT n,m;"■"
5Ø NEXT n
6Ø GOTO 1Ø
```

```
8 MODE 2
9 VDU23;82Ø2;Ø;Ø;Ø;
1Ø FOR row=Ø TO 3Ø
2Ø LET column=RND(19)
3Ø COLOUR 128+RND(7)
4Ø PRINT TAB(column,row)"□";
5Ø NEXT row
6Ø GOTO 1Ø
```

```
7 CLSØ
1Ø FOR N=Ø TO 63
2Ø LET M=RND(32)−1
3Ø LET C=RND(9)−1
4Ø SET (N,M,C)
5Ø NEXT N
6Ø GOTO 1Ø
```

Here, Line 1Ø sets the depth of the pattern you are going to print on the screen, and tells the computer to print it out one line at a time.

Line 2Ø sets the width of the overall pattern, and—with Line 4Ø—tells the computer to print little squares randomly across that width.

Line 3Ø randomizes the colours in the little squares.

```
1Ø PRINT "♥"
15 FOR N=Ø TO 24
2Ø M=INT(RND(1)*16)
25 C=INT(RND(1)*4Ø)
3Ø POKE 1Ø24+(4Ø*N)+C,16Ø
4Ø POKE 55296+(4Ø*N)+C,M
5Ø NEXT
6Ø GOTO 15
```

The Commodore program works in a slightly different way from those on the other machines. Line 15 counts off the lines of the screen, from top to bottom. Line 25 selects squares randomly across

the width of each screen line in turn, while Line 2Ø chooses random colours to print in each square.

VARIATIONS ON A THEME

Trying variations on this theme will help make you familiar with both the FOR . . . NEXT Statement and the RND function.

Here are two for each machine. Don't forget the NEW between them.

```
1Ø FOR n=Ø TO 21
2Ø FOR m=Ø TO 31
3Ø INK RND*7+1
4Ø PRINT AT n,m;"■"
45 NEXT m
5Ø NEXT n
6Ø GOTO 1Ø
```

```
1Ø LET n=RND*21
2Ø FOR m=Ø TO 31
3Ø INK RND*7+1
4Ø PRINT AT n,m;"■"
45 NEXT m
6Ø GOTO 1Ø
```

```
8 MODE 2
9 VDU 23;82Ø2;Ø;Ø;Ø;
1Ø FOR row=Ø TO 3Ø
2Ø FOR column=Ø TO 19
3Ø COLOUR 128+RND(7)
4Ø PRINT TAB(column,row)"□";
45 NEXT column
5Ø NEXT row
6Ø GOTO 1Ø
```

```
8 MODE 2
9 VDU 23;82Ø2;Ø;Ø;Ø;
1Ø LET row=RND(31)-1
2Ø FOR column=Ø TO 19
3Ø COLOUR 128+RND(7)
4Ø PRINT TAB(column,row)"□";
5Ø NEXT column
6Ø GOTO 1Ø
```

```
8 CLSØ
1Ø FOR N=1 TO 6Ø
2Ø FOR M=Ø TO 31
3Ø LET C=RND(9)-1
4Ø SET (N,M,C)
45 NEXT M
5Ø NEXT N
6Ø GOTO 1Ø
```

```
8 CLSØ
1Ø LET N=RND(6Ø)
2Ø FOR M=Ø TO 31
3Ø LET C=RND(9)-1
4Ø SET (N,M,C)
45 NEXT M
6Ø GOTO 1Ø
```

```
1Ø PRINT "♥"
2Ø FOR M=Ø TO 999
3Ø LET C=INT(RND(1)*16)
4Ø POKE 1Ø24+M,16Ø
5Ø POKE 55296+M,C
6Ø NEXT M
7Ø GOTO 2Ø
```

```
1Ø PRINT "♥"
2Ø LET N=INT(RND(1)*25)*4Ø
3Ø FOR M=N TO N+39
4Ø LET C=INT(RND(1)*16)
5Ø POKE 1Ø24+M,16Ø
6Ø POKE 55296+M,C
7Ø NEXT M
8Ø GOTO 2Ø
```

Do keywords such as PRINT always have to be entered in capital letters?

Yes — except on Commodore, where you must use *capitals* if you are in *upper case mode*, but *small letters* if you are in *text mode*.

Before you go any further, here is a small experiment you should try—on the Spectrum, Acorn, Dragon and Tandy only. Delete Line 45 from the first of the two programs, and RUN it again with this line substituted:

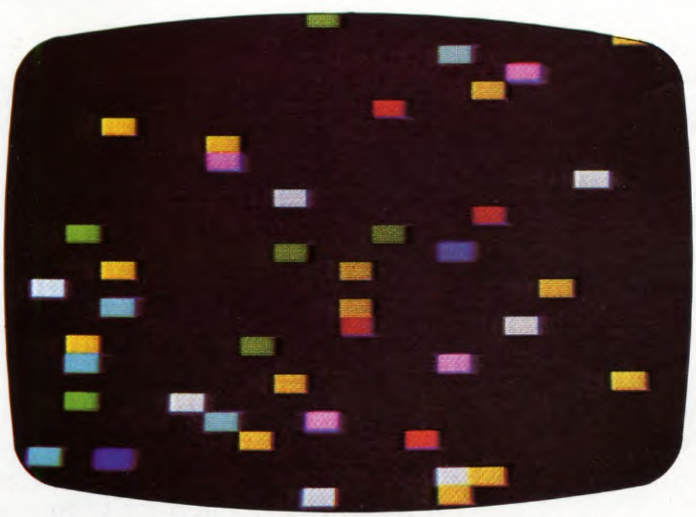
```
55 NEXT m
```

```
55 NEXT column
```

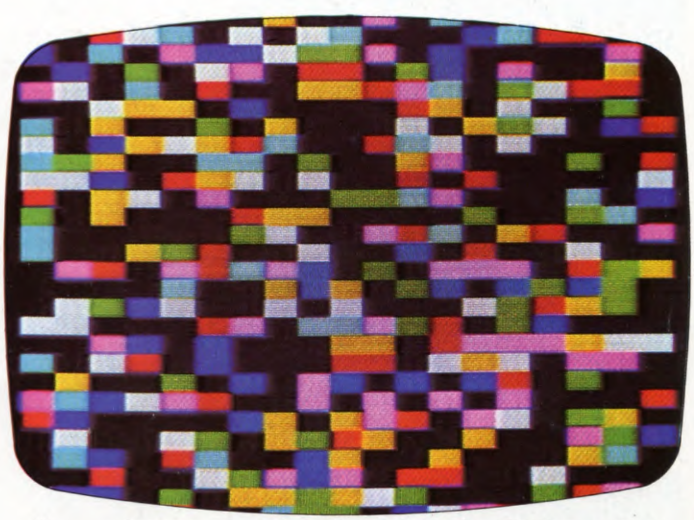
```
55 NEXT M
```

You have just discovered the final important fact about FOR . . . NEXT loops: when you have two such loops in the same program, one must be 'nested' *completely inside* the other or be completely separate. If they overlap, your program will not work.

On the Commodore the problem does not arise on this occasion. But the principle—that one loop must be *completely inside* the other—is the same.



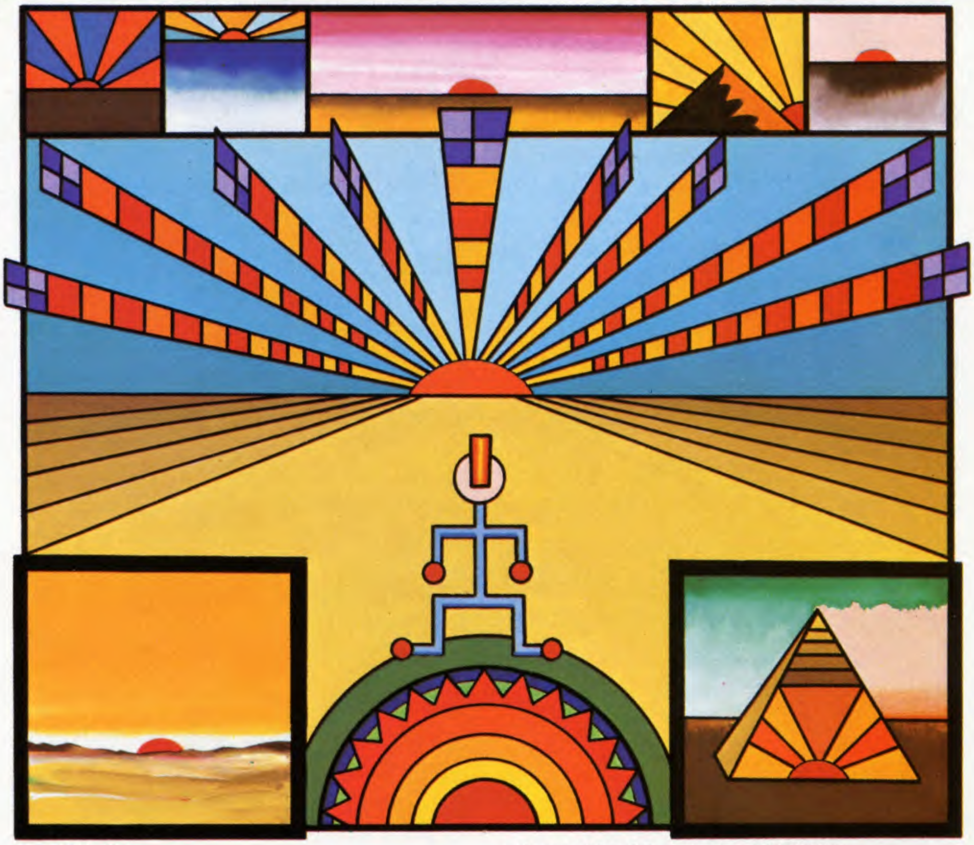
1. First the 'patchwork' builds up on the screen



2. . . . then repeats itself. This is the BBC version

SUNSET PATTERN

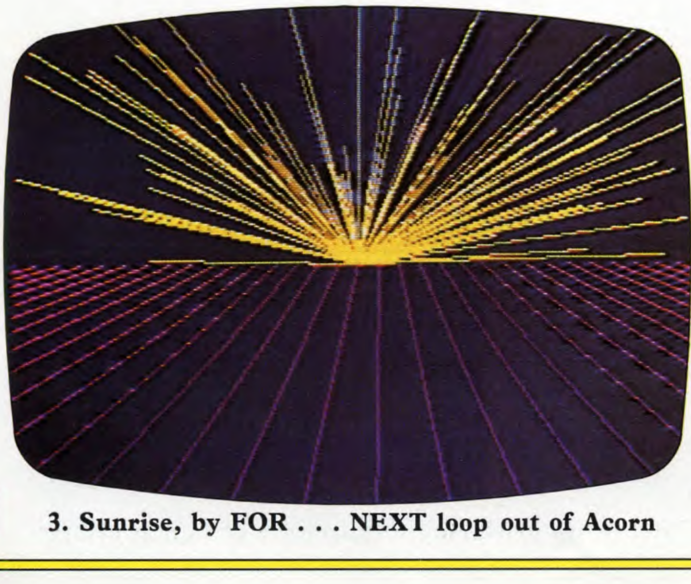
This program uses a FOR . . . NEXT loop to create a 'sunset' pattern. The first part of the program fixes a point in mid-screen, and draws from there to random points across the top of the screen. The second part draws perspective lines at the bottom half of the screen, starting from a series of fixed points.



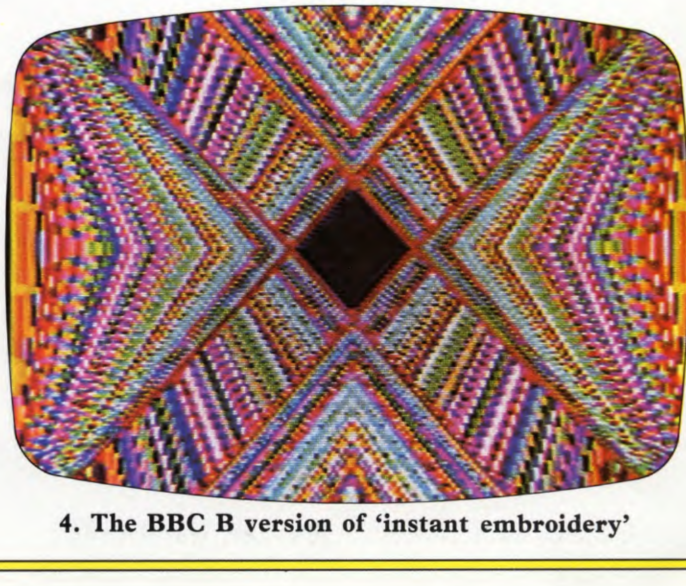
```
5 BORDER Ø:PAPER Ø:INK 6:CLS
1Ø FOR n=1 TO 8Ø
2Ø PLOT 127,75
3Ø DRAW INT (RND*25Ø) —125, INT(RND*97)
4Ø NEXT n
45 INK 5
5Ø FOR n=75 TO Ø STEP —15
6Ø PLOT 127, 75
7Ø DRAW —127,—n: PLOT 127,75:
   DRAW 127,—n
8Ø NEXT n
1ØØ FOR n=—127 TO 127 STEP 2Ø
11Ø PLOT 127,75
12Ø DRAW n,—75
13Ø NEXT n
```

```
1Ø MODE1
15 GCOLØ, 2
2Ø FOR S=1 TO 8Ø
3Ø LET X=RND(128Ø)
4Ø LET Y=512+RND(512)
5Ø MOVE 64Ø,512
6Ø DRAW X,Y
7Ø NEXT S
75 GCOLØ, 1
8Ø FOR L=Ø TO 128Ø STEP 4Ø
9Ø MOVE L,512
1ØØ DRAW(L—512)*4,Ø
11Ø NEXT L
```

```
2Ø PMODE 3,1
3Ø PCLS 3
4Ø COLOR 2
5Ø SCREEN 1,Ø
6Ø FOR N=1 TO 8Ø
7Ø LINE(127,95)—(256—RND(256),
   96—RND(96)),PSET
8Ø NEXT N
9Ø COLOR 4
1ØØ FOR N=95 TO 191 STEP 12
11Ø LINE(127,95)—(Ø,N),PSET
12Ø LINE(127,95)—(255,N),PSET
13Ø NEXT N
14Ø FOR N=Ø TO 255 STEP 1Ø
15Ø LINE (127,95)—(N,191), PSET
16Ø NEXT N
17Ø GOTO 17Ø
```



3. Sunrise, by FOR . . . NEXT loop out of Acorn



4. The BBC B version of 'instant embroidery'

INSTANT EMBROIDERY

Finally, here is a really spectacular program which works on four of our machines:

S

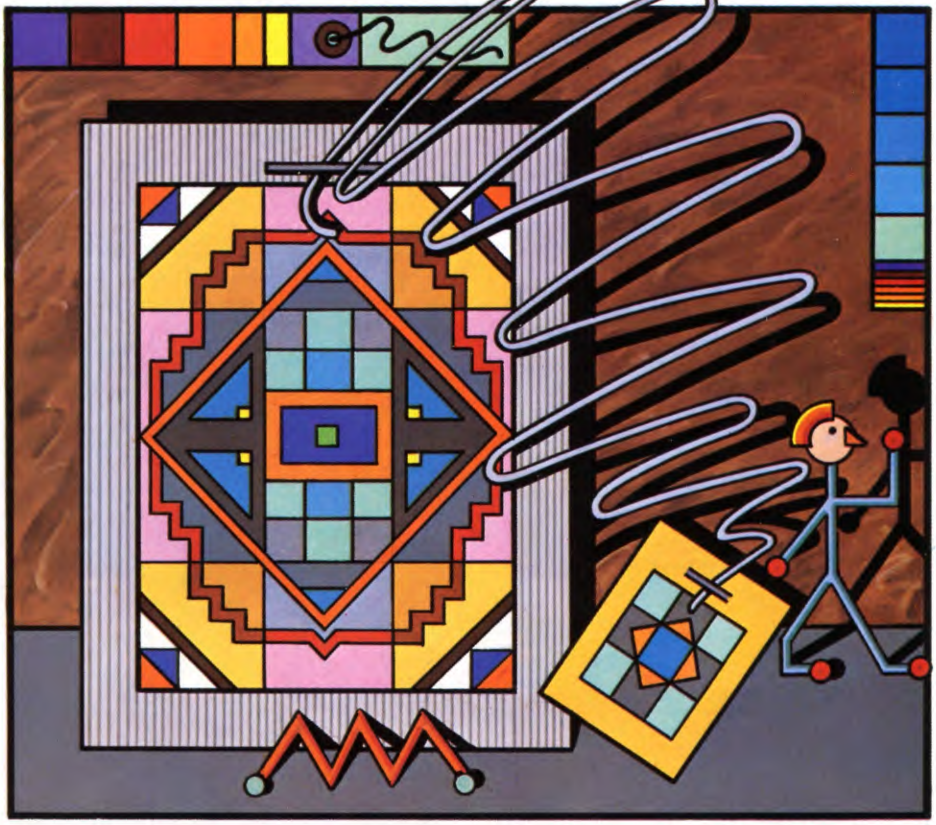
```
1Ø FOR n=Ø TO 255 STEP 2
15 INK RND*8
2Ø PLOT Ø,Ø: DRAW n,175
3Ø PLOT 255,Ø: DRAW —n,175
4Ø PLOT Ø,175: DRAW n, —175
5Ø PLOT 255,175: DRAW —n,—175
6Ø NEXT n
7Ø GOTO 1Ø
```

B

```
8 MODE 2
1Ø FOR N=Ø TO 1279 STEP 1Ø
15 GCOL Ø,RND(7)
2Ø MOVE Ø,Ø: DRAW N,1Ø23
3Ø MOVE 1279,Ø: DRAW 1279—N,1Ø23
4Ø MOVE Ø,1Ø23: DRAW N,Ø
5Ø MOVE 1279,1Ø23: DRAW 1279—N,Ø
6Ø NEXT N
7Ø GOTO 1Ø
```

DT

```
3 PMODE 3,1
6 PCLS
9 SCREEN1,Ø
1Ø FOR L=Ø TO 255 STEP 2
15 COLOR RND(4)
2Ø LINE (Ø,Ø)—(L,191),PSET
3Ø LINE (255,Ø)—(255—L,191),PSET
4Ø LINE(Ø,191)—(L,Ø),PSET
5Ø LINE(255,191)—(255—L,Ø),PSET
6Ø NEXT L
7Ø GOTO 1Ø
```

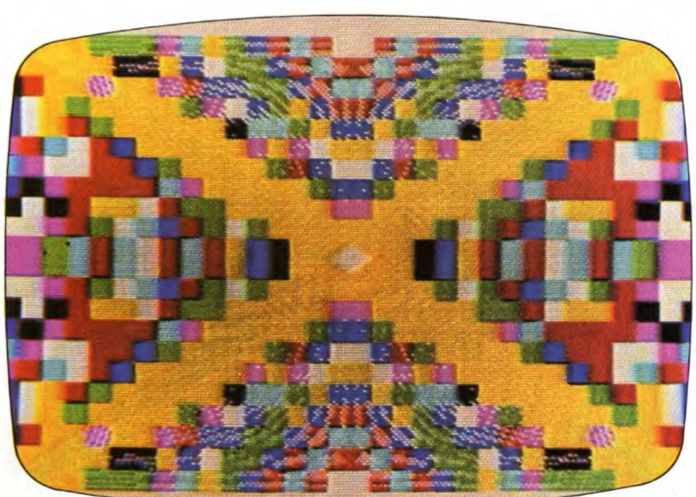


Each of the four segments of this pattern begins with a dot in one corner of the screen. What the FOR . . . NEXT loop is doing is to count across the opposite side of the screen while a pattern of lines is drawn between the pairs of points thus created.

Exactly how the graphics work is in a later chapter. But try deleting Lines 3Ø and 4Ø and you will get the general idea.

You can experiment, too — by deleting some of the 'drawing' lines, by varying the colours, by lengthening the STEPs in Line 1Ø and by excluding the GOTO line at the end. In minutes you can create hundreds of different patterns — 'instant embroidery,' in fact.

In doing this, you are not just creating pretty pictures on the screen. You are also helping to familiarize yourself with one of the most useful 'tools' available to the programmer.



5. Spectrum's version — stronger, less delicate,



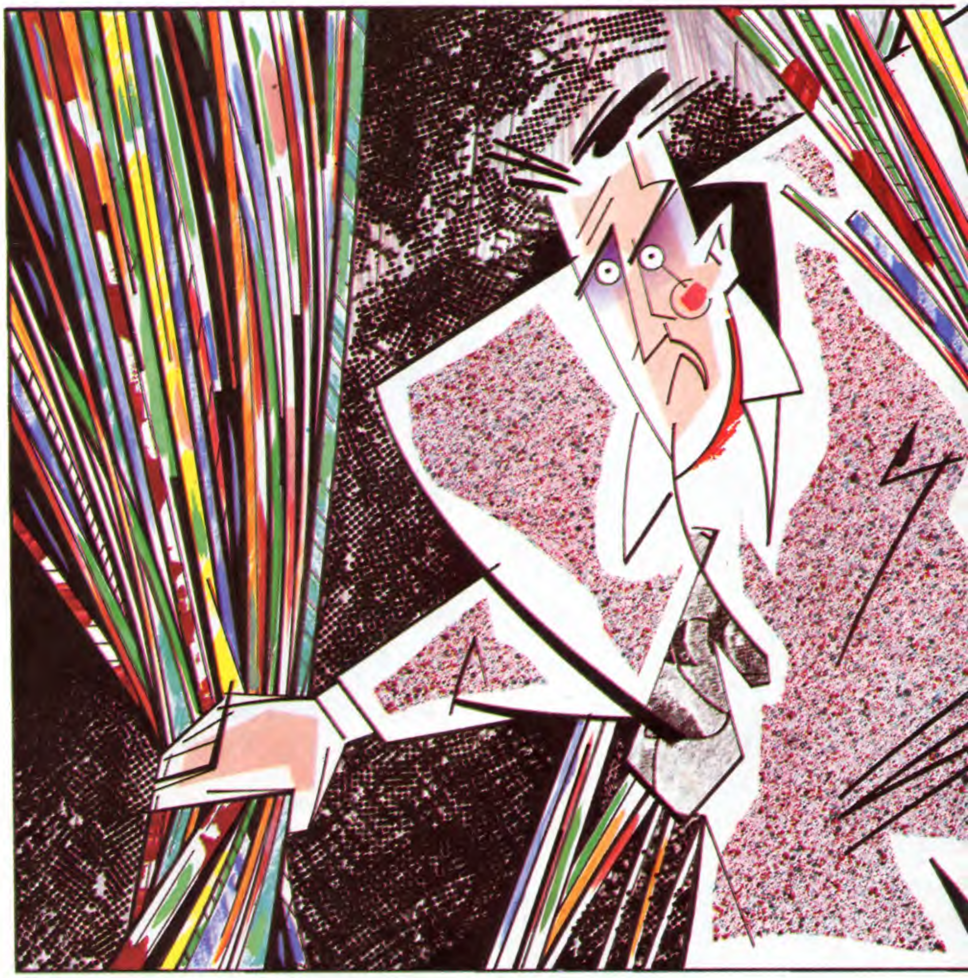
6. . . . but producing a constantly-changing pattern

UNTANGLING YOUR SAVES AND LOADS

Nothing is more infuriating than a program which won't SAVE or a game that won't LOAD. Here's how to get the frustration down to an irreducible minimum

Few things bring a newcomer down to the harsh reality of computers' exacting demands more quickly than early problems of SAVEing and LOADing programs with a cassette recorder.

Almost everyone has tape problems. The causes are not always easy to isolate—even when you do know your way round a computer. Nor is it possible to completely eliminate SAVE and LOAD errors by adopting apparently fail-safe recording and playback procedures. But you can establish routines which will minimize the risks.



CORRECT SET-UP

Some computers use a purpose-built tape recorder which makes a single direct connection to the host machine and there should be no problem linking this up. But most computers can be linked to any cassette-type tape recorder — often using more than one type of lead combination. Typically the lead consists of a DIN plug connected to three jack plugs, but can be a DIN plug to a DIN plug with or without a jack plug.

If you buy a lead—or make one up from existing audio leads you may have — do ensure it is suitable: LOADing difficulties will be evident right away if it is not.

CONTROL SETTINGS

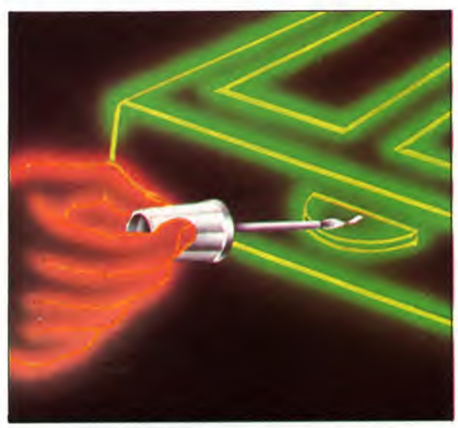
An early — and important — step which prevents a hit-or-miss approach to SAVE and LOAD routines is to establish exact settings of the tone, volume and feature controls on the recorder you are using. Some computer instructions provide precise directions on the best way of doing this—advice which you should follow carefully. Others leave it all up to you. If you are using a special data cassette recorder, follow the specific setting-up instructions which should be provided.

With an ordinary audio recorder, start by switching out all the special features such as noise reduction systems and filters. Set the tone control(s) to give maximum treble ('high') and leave the recorder in this state whenever it is used with the computer. Switch out special filters.

Choose a mid-point volume setting and try LOADing a prerecorded program, such as on the demonstration tape which came with your machine. If the program fails to LOAD, try increasing the volume by a little and running it again. If the program still fails to LOAD, continue increasing the volume, step-by-step, running the program tape again each time.

If you reach the maximum setting with no success, reset the volume control to slightly less than the original mid-point position and then progressively decrease the volume setting, running the tape again each time.

If for some reason the program fails to LOAD at any volume setting, check the con-



1. If you can, permanently mark the best volume setting

- GETTING THE RIGHT SET-UP
- CONTROL SETTINGS
- THE **SAVE** COMMAND
- VERIFYING YOUR SAVES
- THE **LOAD** COMMAND



nections again. Try another lead or program tape, or borrow a tape recorder known to work with your make of computer. If nothing works ask your dealer's advice—it could very easily be a fault with your computer.

When you have found a successful volume setting for LOADing, note the control setting, establish upper and lower limits, then mark the mid-point for future use. You can usually leave the volume control at the same setting for SAVEing operations — many recorders have automatic recording level control in any case.

To test that the volume levels are right for SAVEing, clear the computer memory (type NEW, then press ENTER or RETURN or simply switch the machine off and then back on) and enter this dummy program:

```
10 REM TEST
20 REM
30 REM
40 REM
```

and then follow the SAVE command routine used by your computer. When you LOAD and LIST, you should get the program back.

THE **SAVE** COMMAND

The exact form of SAVE command used with your computer is given in your manual. But several preliminary steps are needed.

Before SAVEing any program, load your recorder with a suitable cassette and wind it just past the point where the leader joins the tape.

Next, decide on a suitable *file name* for your program. All data is stored in the form of 'files' of one type or another, regardless of the actual storage methods used. But, for recordings made on tape, particular types of file names are used with the SAVE command.

The file or program name can usually be any combination of characters or symbols within the line length specified for your machine. On most machines the file name is restricted to ten characters or less; only a few permit anything longer.

You can call your program anything you like within the allowable line length and short abbreviations are just as good as long-winded descriptions. But everything that forms part of the file or program name must be enclosed wholly by quote marks before it can be recognised as such by the computer:

```
SAVE "PROGNAME01"
SAVE "ProgName01"
```

Both these are valid SAVE commands—but note the use of both capitals and lower case letters in the second example. If you mix capitals and lower case in this way on a program name you must repeat the name *exactly* when you ask the computer to LOAD the program.

TROUBLESHOOTER

- Use a simple, reliable mono portable recorder and, if you can, reserve this for exclusive use with the computer. Always use mains power to ensure constant motor speeds
- Avoid using sophisticated stereo recorders unless its facilities can be switched out and mono playback is possible
- Use good quality audio or data tape —if you find a brand which is reliable, stick to it. A popular or much-worked-on program will be LOADed many times and poor quality tapes will not put up with this treatment
- Watch the screen for specific instructions — you must leave the recorder in 'play' mode until LOADing is complete
- Try alternative volume, tone and feature control settings on your recorder if LOADing proves unsuccessful
- Move the recorder away from the TV/monitor if a program won't LOAD but has LOADed successfully before. Try another program recording to see if it works — if it does, the first tape may have been damaged in some way
- If volume settings need frequent adjusting from program to program, mark the necessary volume settings on the label of the tape cassette
- Make a habit of rewinding tapes after they have been used. This may prevent you later trying to LOAD from blank parts of the tape. Start program searches at the beginning of a tape rather than put too much reliance on the tape counter
- Store your program tapes in a dry, dust-free place, away from electrical appliances and heat
- Make sure your new program LOAD doesn't conflict with something already in memory

Q+A

Although my friends with expensive cassette recorders make a great thing about cleaning their machines, is it really necessary just for data recording?

Perhaps more so. Regular maintenance of the tape recorder—and its leads—takes only a few minutes yet can save the frustration of poor SAVEs and LOADs.

Use a good quality cassette recorder cleaning kit to remove oxide deposits from the drive mechanism and head(s) of your recorder. Be especially careful not to scratch the delicate head. Avoid re-using 'bud' type wipes, though these may be used with caution when new to remove stubborn build ups which cannot be handled by a play or two of the cleaning tape. If you use cleaning solution make sure it has an alcohol—not solvent—base.

If you want visual spacing, consider using punctuation marks (other than full quotes) instead:

```
SAVE "PROG.NAME"
SAVE "PROG/NAME"
SAVE "PROG(NAME)"
```

If your computer is linked to a remote control facility on the recorder, set the controls to 'record'. Enter your selected program name after your computer's tape SAVE command and then hit ENTER or RETURN. The computer should then take control of the recorder motor until the SAVE routine is completed.

If control is left to you, enter the SAVE command, put the recorder into 'record' mode and then press ENTER or RETURN. Wait until the 'ready' prompt shows on the screen display.

If you are storing a particularly important program, repeat the SAVE routine for at least one backup copy – but first make sure there is room on the tape to do this. Better still, record the backup copy on a quite separate tape.

The time taken for the program to load depends on two factors: its size, in terms of memory usage, and the speed of data flow between the computer and recorder, which is fixed on some machines and selectable on others.

The most reliable recordings are made at the slowest rates and many computers will use their own minimum settings unless instructed otherwise. Fast settings use less tape and obviously SAVE and LOAD programs more quickly. With the very fastest rates it is important to use top-quality tapes and a proven recorder.

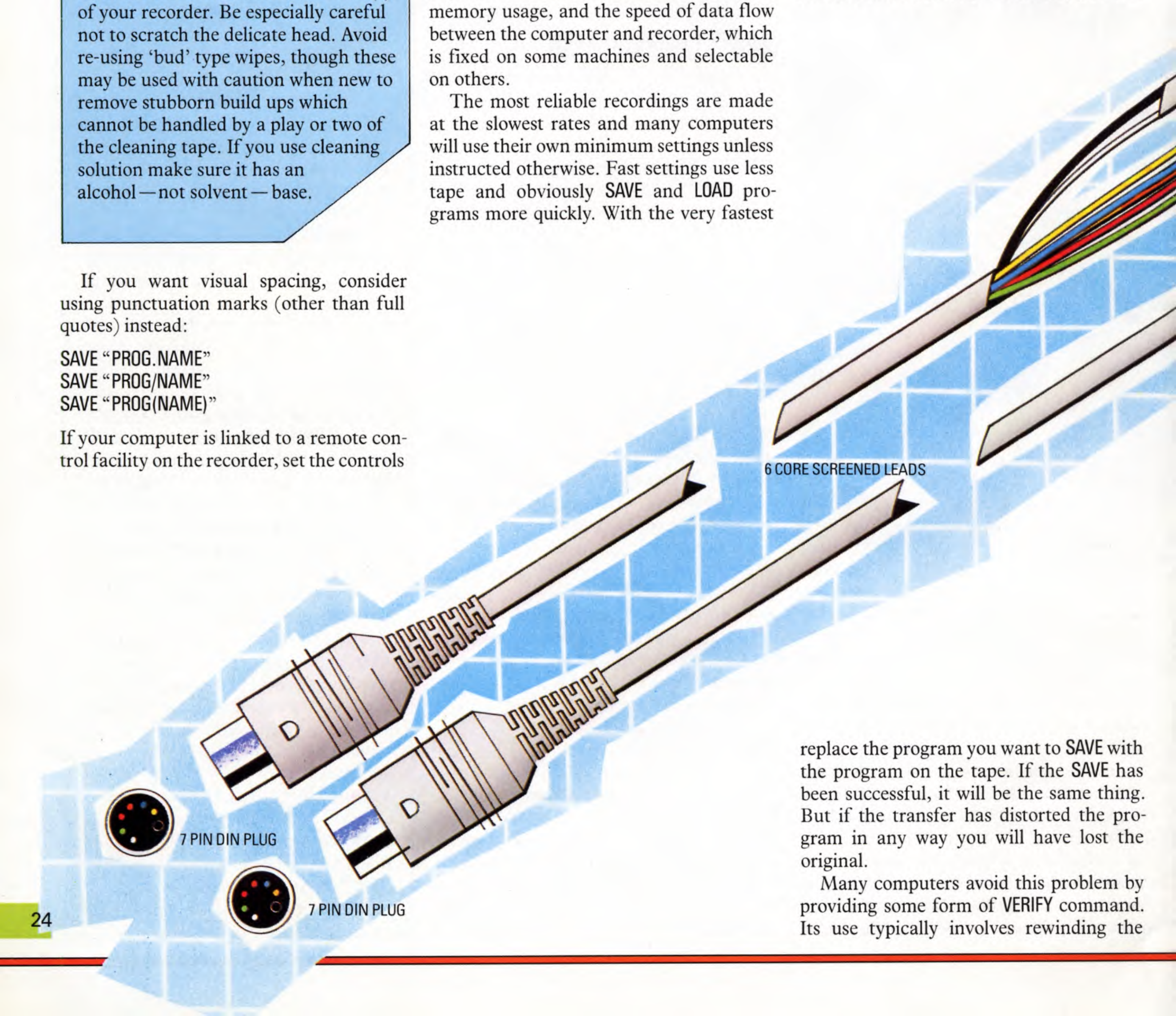
Other variations of the SAVE command to deal with special applications are explained in a later article.

Make a habit of labelling the cassette as soon as you have made a new program recording. Small, self-adhesive labels are particularly convenient for this. Remove the tab from the back of the cassette to prevent accidental erasing or overwriting of an important program.

VERIFYING YOUR SAVES

One way of checking whether or not a program has been SAVEd properly is to LOAD and RUN it. But this could mean the loss of the program if the SAVE has not been successful. In the computer's memory you will replace the program you want to SAVE with the program on the tape. If the SAVE has been successful, it will be the same thing. But if the transfer has distorted the program in any way you will have lost the original.

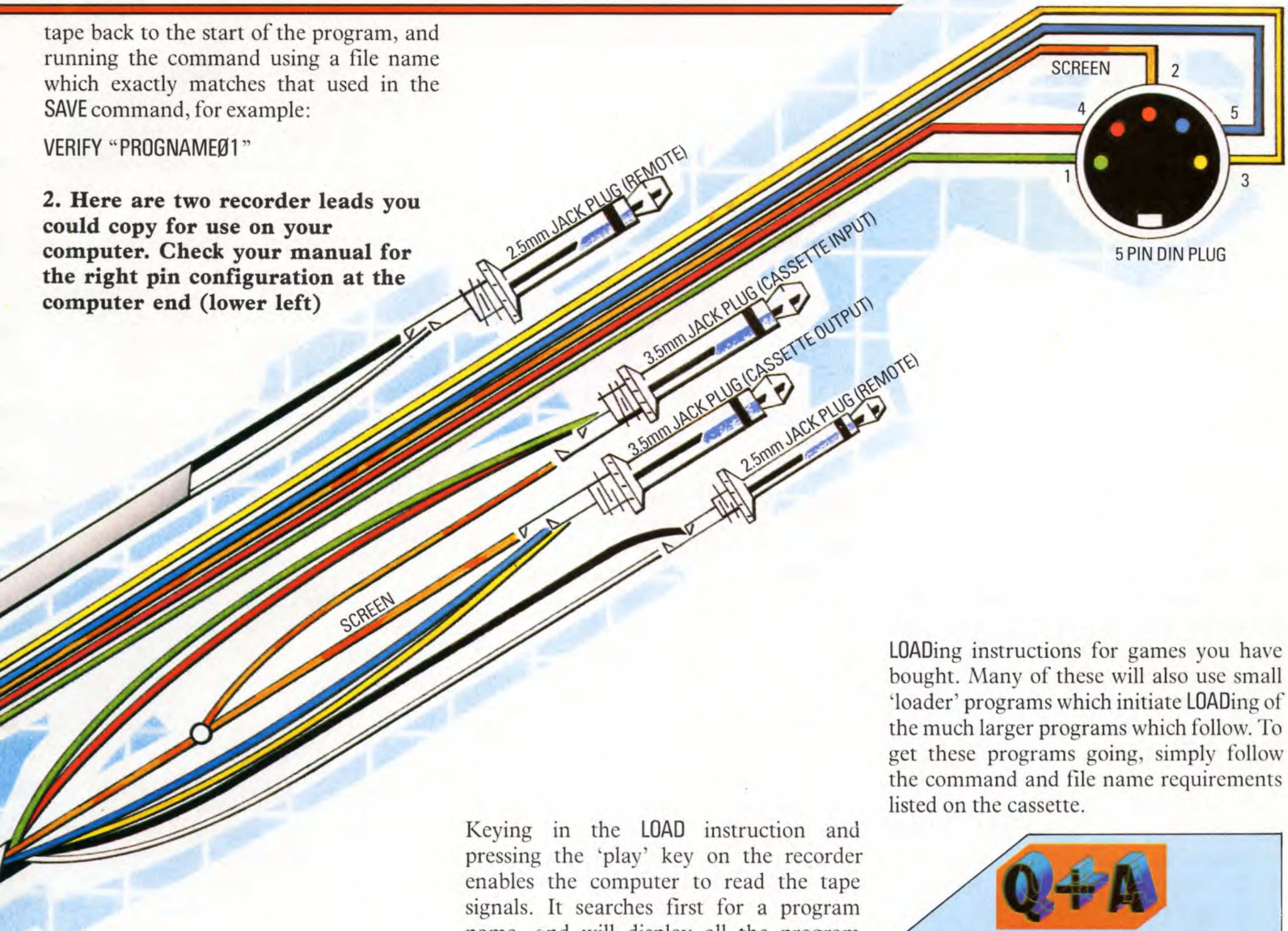
Many computers avoid this problem by providing some form of VERIFY command. Its use typically involves rewinding the



tape back to the start of the program, and running the command using a file name which exactly matches that used in the SAVE command, for example:

VERIFY "PROGNAMEØ1"

2. Here are two recorder leads you could copy for use on your computer. Check your manual for the right pin configuration at the computer end (lower left)



The computer will then read through the SAVEd program and check it against the one it has in its memory. If the program has corrupted during recording an error message will be displayed on the screen. This gives you the opportunity to try the SAVE again until you are successful.

THE LOAD COMMAND

If you start by using prerecorded program tapes, your first problems are likely to be with the LOAD command. As with the SAVE command, there can be several types of LOAD command even for a single type of computer. But for the moment look at the simplest form:

LOAD "PROGNAMEØ1"

This exactly duplicates the form of file address used by the SAVE and VERIFY commands – the program will not load if there is any error in the program file name. Keying in the LOAD instruction and pressing the 'play' key on the recorder enables the computer to read the tape signals. It searches first for a program name, and will display all the program names until it reaches the one which matches the file name you have entered after the LOAD command. When this name is found, the data which follows is automatically LOADed into the memory.

If the recorder is remotely controlled by the computer, the cassette drive automatically stops after loading, but it is up to you to press the 'stop' key. If your recorder is not controlled by the computer, press 'stop' when prompted to do so.

The program you LOAD will replace whatever is already in your computer's available RAM memory – so if you have an important program in there, make sure you have SAVEd it beforehand.

Wind the four-line test program back to its start and try to LOAD it using the chosen file name. If you have problems, refer to the troubleshooting guide here.

Other forms of LOAD command are needed for accessing machine code data, for program relocation, and for merging one program with another. Your first encounter with these may be as part of the LOADing instructions for games you have bought. Many of these will also use small 'loader' programs which initiate LOADing of the much larger programs which follow. To get these programs going, simply follow the command and file name requirements listed on the cassette.

Q+A

Can I use old music cassettes for recording program data?

Although it is best to use good quality data tapes, perfectly acceptable results are possible using top-quality audio cassettes. If by 'old music cassettes' you mean tapes which have lain around on a parcel shelf of a car or in some equally unsatisfactory storage location, you are asking for trouble.

Tapes which have shed minute fragments of their coating or bind slightly may make no appreciable difference if you are playing music—but can wreak havoc on data SAVEing or LOADing. Only one item of information has to go astray for a SAVE or LOAD error.

A further problem of using tapes previously used to record data or music is the continued presence of part of the original signal after the new data recording has been made. This can easily spoil it.

GET MOVING ON ANIMATION

Want to bring your games programming to life? Then start with these simple graphics characters—you can produce them from the ROM graphics on your computer

Playing commercial computer games is fine up to a point, but the time comes when most people feel the need to let loose their imaginations and create their own games.

Games programming is not easy; you have to start with simple things and build up. But this helps you to learn to think logically and increases your programming skill. You'll also have more fun.

In games programming the first thing you need to learn, apart from BASIC skills, is the technique of animation.

To create the illusion of movement, the computer programmer uses much the same techniques as the cartoonist who animates a movie. He creates two (or more) pictures and alternates them rapidly — ideally, about 24 times a second.

But there is an important difference. In movie animation, the cartoonist can rely on the film projector to get rid of each picture when it is no longer wanted. In computer animation this is not so. Unless you do something about it, any segment of a picture which you 'project' onto a given area of the screen will stay there.

One way of getting rid of the unwanted segment is simply to print something over it. The computer cannot put two images on the same screen location at the same time.

So if, for example, Line 1Ø tells the computer to print an A at a particular location, any later line which prints (say) B at the same location will dispose of the A as well.

The programs below include several examples of this kind of substitution.

But what if you have nothing that you want to print over the unwanted character? Then you must remember to include in a later line an instruction to print a space, "□", at the relevant screen location.

Forget this detail, and your screen can easily be cluttered with unwanted bits of arms, legs and bodies.

There are big differences between one computer and another in the way you get graphics characters on the screen. The standard, or ROM, graphics characters vary. So does the way you PRINT them on the screen. And does the way you make them move.

A simple animation you can try on the Dragon and Tandy is the small 'creepy crawly' (below). To create him, program the machine as follows:



```
1Ø PRINT @ 238, "OOO"
2Ø PRINT @ 2Ø6, ")))"
3Ø PRINT @ 241, "<"
4Ø PRINT @ 27Ø, ")))"
```

RUNning this program is rather an elaborate way to produce a simple image, but it does illustrate some interesting points:

First, it gives you some feel for the relative positions of the screen locations. The middle of the insect is about in the middle of the screen, at location 239.

Second, it shows what happens when the machine is told to PRINT more than one character at one screen location—it simply carries on and prints the characters at the adjoining (higher numbered) locations. That is why the 'feelers' in line 3Ø are at 241. Locations 238-24Ø are already occupied by the insect's body.

A more convenient way of setting up the insect is to amalgamate the program above into a single line. Incidentally, if you have not discovered it yet, on the Dragon there is a shorthand for PRINT — just use the character "?". When the program is listed the machine will show PRINT instead of ?.

The simplified program looks like this:

```
1Ø PRINT @ 238, "OOO<":PRINT @ 2Ø6,
   ")))":PRINT @ 27Ø, ")))"
```

If you add two more lines you get some animation:

```
2Ø PRINT @ 238, "OOO<":PRINT @ 2Ø6,
   "(((":PRINT @ 27Ø, "((("
3Ø GOTO 1Ø
```

RUNning the program produces a blur and does not look like animation at all. The reason for this is that the pictures are being swapped too quickly. If you insert a FOR...NEXT loop it makes the computer pause and count, and animation is clearer. Any number can be used in the FOR...NEXT loop lines (Lines 15 and 25), not only 15. Other numbers will simply give you a longer or shorter delay.

Try adding these extra lines:

```
15 FOR L=1 TO 15
17 NEXT L
25 FOR L=1 TO 15
27 NEXT L
```

Now you have a little insect which flails about uselessly but, nonetheless, gives believable animation. Movement comes a little later (see **Movement**).



LOW-RESOLUTION GRAPHICS

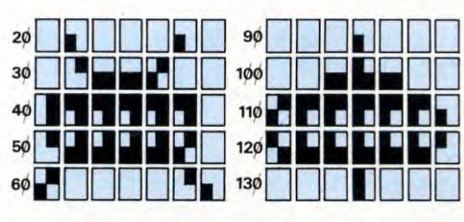
You can use the computer's low-resolution graphics to create more interesting animations. The User's manual shows the graphics characters which you can use. Each graphics character has a code (a value from 128 to 143). To display them on the screen you use CHR$ then the code of the graphics character in brackets.

For example, to display the character represented by code 138 towards the middle of the screen you would type:

```
1Ø PRINT @ 239, CHR$ (138)
```

The following program uses the low-resolution graphics to animate this spinning satellite (**fig. 1** shows two views of it which are alternated in the program).

```
1Ø CLEAR 5ØØ:CLS
2Ø PRINT@ 174,CHR$(143)+CHR$(141):
   PRINT@ 178,CHR$(143)
```



1. How to build your satellite



- THE PRINCIPLES OF ANIMATION
- MOVING GRAPHICS FROM THE 'TYPEWRITER' KEYS
- HOW TO USE ROM GRAPHICS



```
3Ø PRINT@ 2Ø6,CHR$(14Ø)+CHR$(132)+
   CHR$(141)
4Ø PRINT@ 236,CHR$(137)+CHR$(129)+
   CHR$(129)+CHR$(129)+CHR$(129)+
   CHR$(129)+CHR$(141)
5Ø PRINT@ 268,CHR$(135)+CHR$(132)+
   CHR$(132)+CHR$(132)+CHR$(132)+
   CHR$(133)+CHR$(135)
6Ø PRINT@ 3ØØ,CHR$(143):PRINT@
   3Ø3,CHR$(133):PRINT@ 3Ø5,CHR$
   (143)+CHR$(143)
7Ø FOR X=1 TO 1Ø
8Ø NEXT X
9Ø PRINT@ 173,CHR$(141):PRINT@ 177,
   CHR$(141)
1ØØ PRINT@ 2Ø5,CHR$(139)+CHR$(14Ø)+
   CHR$(14Ø)+CHR$(137)
11Ø PRINT@ 236,CHR$(138)+CHR$(13Ø)+
   CHR$(13Ø)+CHR$(13Ø)+CHR$(13Ø)+
   CHR$(13Ø)+CHR$(143)
12Ø PRINT@ 268,CHR$(139)+CHR$(136)+
   CHR$(136)+CHR$(136)+CHR$(136)+
   CHR$(137)+CHR$(143)
13Ø PRINT@ 3ØØ,CHR$(137):PRINT@
   3Ø3,CHR$(143):
   PRINT@ 3Ø5,CHR$(139)
   +CHR$(141)
14Ø FOR X=1 TO 1Ø
15Ø NEXT X
16Ø GOTO 1Ø
```

Do not worry about Line 1Ø in the program. CLEAR 5ØØ simply makes enough memory space available for the strings created by using CHR$ codes and CLS gives you a clear screen for your animation.

Look up the graphics characters represented by the codes after CHR$ in the program. And by referring to fig. 1, try to see how the spaceship is constructed. The program instruction '+CHR$()' means 'PRINT CHR$() on the next screen location.'

MOVEMENT

Here, at last, is a program which not only animates the insect, but also moves it across the screen:

```
1Ø CLS
2Ø FOR N=Ø TO 28
3Ø PRINT@ 192+N,")))":PRINT@ 224+N,
   "OOO<":PRINT@
   256+N,")))"
4Ø FOR X=1 TO 1Ø
5Ø NEXT X
6Ø PRINT@ 192+N,"□□□": PRINT@
   224+N,"□□□□": PRINT@
   256+N,"□□□"
7Ø PRINT@ 192+N,"(((":PRINT@
   224+N,"OOO<":PRINT@ 256+N,"((("
8Ø FOR X=1 TO 1Ø
9Ø NEXT X
1ØØ PRINT@ 192+N,"□□□":PRINT@
   224+N,"□□□□":PRINT@
   256+N,"□□□"
11Ø NEXT N
12Ø GOTO 2Ø
```

There are three FOR...NEXT loops in the program. The two loops using X both do exactly the same thing as in your first attempt at using them — they slow the printing down by causing the computer to count to 1Ø each time.

The FOR...NEXT loop using N has a different function. The loop actually makes the insect move across the screen one location at a time. (For how a FOR...NEXT loop works, see BASIC programming 2.)

You may be puzzled that Line 2Ø reads FOR N=Ø TO 28 when there are 32 spaces (the top line numbered from Ø to 31) available in each line of the screen. The reason for this is that the insect is four spaces long — if you had any more than 28 the insect's antennae would appear at the opposite side of the screen, one line down. This is because of the way the screen locations are numbered. Screen location 32, for example, is the first location on the second row from the top of the screen.

Lines 6Ø and 1ØØ appear to be PRINTing nothing at all. Try deleting them and see what happens, though. These are 'unPRINTing' lines as described earlier.

You can make up bugs and monsters on Acorn machines simply by using the ordinary keyboard characters such as brackets, dashes and letters.

Below is a small 'creepy crawlie' made up this way which is very easy to animate.

To begin with, try creating him in a static position.

```
)))     )))     )))     )))
OOO <   OOO <   OOO <   OOO <
)))     )))     )))     )))
```

```
5 CLS
1Ø PRINT TAB(15,1Ø);"OOO"
2Ø PRINT TAB(15,9);")))"
3Ø PRINT TAB(15,11);")))"
4Ø PRINT TAB(18,1Ø);"<"
```

(Note that when you type in this program you must not leave a space after **TAB**.)

Now **RUN** the program.

This is a very simple picture, and not terribly lifelike, but it does illustrate one or two interesting points.

First, it gives you an idea of the relative positions of the screen locations. The middle of your insect is on Line 1Ø from the top and about halfway across the screen.

Second, it shows the effect of telling the computer to print more than one character on a single location. It simply carries on and prints the extra characters on the adjoining locations. That is why the 'feelers' in Line 4Ø are at 18,1Ø. Locations 15,1Ø;16,1Ø; and 17,1Ø; are already occupied by the insect's body.

To animate the insect you must overprint it with another slightly different one and then swap quickly between the two.

Add these lines to draw the second insect:

```
5Ø PRINT TAB(15,1Ø);"OOO"
6Ø PRINT TAB(15,9);"((("
7Ø PRINT TAB(15,11);"((("
8Ø PRINT TAB(18,1Ø);"<"
```

Add one more line and you have animation:

```
9Ø GOTO 1Ø
```

When you run this new program you may find the flashing cursor spoils the picture. You can get rid of it by adding:

```
7 VDU 23;82Ø2;Ø;Ø;Ø;
```

You may also want to change the speed at which the legs move. You can slow them down by using **INKEY** which tells the computer to wait for a while. The delay is measured in hundredths of a second, so **A=INKEY(1ØØ)** means a delay of one second.

Try adding these lines:

```
45 LET A=INKEY(5Ø)
85 LET A=INKEY(5Ø)
```

You can adjust the speed of the insect's legs by varying the **INKEY** number until you get realistic movement.

USING TELETEXT GRAPHICS

If you have a BBC computer you can make better-looking characters using the computer's graphics set. (Making graphics characters on the Electron requires a different procedure, and is covered in a later article.)

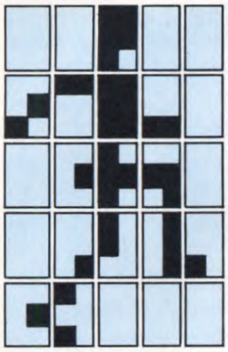
In **MODE 7** the BBC computer can produce a whole range of block graphics which you can build up into interesting shapes.

First draw out a shape on graph paper and divide it into rectangles 2 squares across by 3 squares down. Each of these is a graphics character. Each character has a number which is given in the back of the manual.

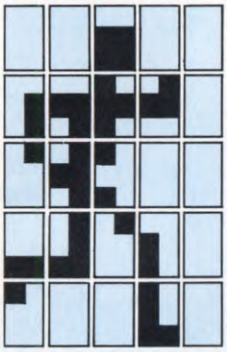
If you look up the numbers and write them out as shown in Fig 2, it is easy to write a program to draw the picture on the screen. Here is one example:

```
1Ø MODE 7
2Ø LET Y=1Ø: LET X=15
4Ø VDU 31,X,Y,146,16Ø,16Ø,191,16Ø,16Ø
5Ø VDU 31,X,Y+1,146,184,163,255,24Ø,16Ø
6Ø VDU 31,X,Y+2,146,16Ø,168,189,236,16Ø
7Ø VDU 31,X,Y+3,146,16Ø,224,165,234,176
8Ø VDU 31,X,Y+4,146,168,177,16Ø,16Ø,16Ø
9Ø VDU 31,X,Y,146,16Ø,16Ø,252,16Ø,16Ø
1ØØ VDU 31,X,Y+1,146,232,236,189,174,16Ø
11Ø VDU 31,X,Y+2,146,162,238,177,16Ø,16Ø
12Ø VDU 31,X,Y+3,146,24Ø,25Ø,162,18Ø,16Ø
13Ø VDU 31,X,Y+4,146,161,16Ø,16Ø,245,16Ø
14Ø GOTO 4Ø
```





160	160	191	160	160
184	163	255	240	160
160	168	189	236	160
160	224	165	234	176
168	177	160	160	160



160	160	252	160	160
232	236	189	174	160
162	238	177	160	160
240	250	162	180	160
161	160	160	245	160

2. The running man — two views

The first line puts the computer into Mode 7, the Teletext mode. Line 2Ø sets the man's position on the screen.

Lines 4Ø to 8Ø define the man in the first position, while Lines 9Ø to 13Ø define him in the second position. Again you may need to use INKEY after Lines 8Ø and 13Ø.

The VDU command controls the screen, so that VDU 31,X,Y means PRINT TAB(X,Y). The next number produces coloured graphics, 145 to 151 making all the different colours. In this case, 146 draws the man in green. The other numbers (16Ø and over) are the ones that control the shape.

MOVEMENT

Making the man move across the screen is also quite easy. All you do is start him in position X=Ø at the left of the screen and move him one position at a time to the other side.

The computer does this by using a FOR...NEXT loop. So you will need to type in two new lines:

```
3Ø FØR X=Ø TO 35
14Ø NEXT X
```

This prints the man at each position from column Ø to column 35.

You may wonder why X only goes up to 35 when there are 4Ø columns on the screen (from X=Ø TO 39). The reason is that the man is five columns wide. If he went past the end of the screen, bits of him would start appearing on the left-hand side. Try changing Line 3Ø and see what happens.

Now add two more lines to introduce a slight delay:

```
85 LET A=INKEY(3)
135 LET A=INKEY(3)
```

Finally, add one more line to turn off the flashing cursor:

```
15 VDU 23;82Ø2;Ø;Ø;Ø;
```

Here, then, is the complete program:

```
1Ø MODE7
15 VDU 23;82Ø2;Ø;Ø;Ø;
2Ø LET Y=1Ø
3Ø FOR X=1 TO 35
4Ø VDU 31,X,Y,146,16Ø,16Ø,191,16Ø,16Ø
5Ø VDU 31,X,Y+1,146,184,163,255,240,16Ø
6Ø VDU 31,X,Y+2,146,16Ø,168,189,236,16Ø
7Ø VDU 31,X,Y+3,146,16Ø,224,165,234,176
8Ø VDU 31,X,Y+4,146,168,177,16Ø,16Ø,16Ø
85 LET A=INKEY(3)
9Ø VDU 31,X,Y,146,16Ø,16Ø,252,16Ø,16Ø
1ØØ VDU 31,X,Y+1,146,232,236,189,174,16Ø
11Ø VDU 31,X,Y+2,146,162,238,177,16Ø,16Ø
12Ø VDU 31,X,Y+3,146,24Ø,25Ø,162,18Ø,16Ø
13Ø VDU 31,X,Y+4,146,161,16Ø,16Ø,245,16Ø
135 LET A=INKEY(3)
14Ø NEXT X
```

Notice that in this case there is no need for any 'un-PRINTing' spaces. The control code 146 acts as a blank space which automatically erases the previous image as the man is moved along.

You would need them, though, if you wanted to make the 'creepy crawlie' move, so spaces are inserted in front of each image as shown in the program below:

```
1Ø CLS
2Ø VDU 23;82Ø2;Ø;Ø;Ø;
3Ø FOR X=1 TO 35
4Ø PRINT TAB(X,10);"□000<"
5Ø PRINT TAB(X,9);"□)))"
6Ø PRINT TAB(X,11);"□)))"
7Ø A=INKEY(1Ø)
8Ø PRINT TAB(X,1Ø);"□000<"
9Ø PRINT TAB(X,9);"□((("
1ØØ PRINT TAB(X,11);"□((("
11Ø A=INKEY(1Ø)
12Ø NEXT X
```



The Commodore version of the 'creepy crawlie' uses exactly the same typewriter symbols as do other machines, but the method of creating him on the screen is different. Try keying in this program:

```
)))     )))     )))     )))
OOO<    OOO<    OOO<    OOO<
)))     )))     )))     )))
```

```
10 PRINT "♥"
20 PRINT ")))"
30 PRINT "OOO<"
40 PRINT ")))"
50 PRINT "S"
60 PRINT "((("
70 PRINT "OOO<"
80 PRINT "((("
90 GOTO10
```

When you RUN the program you will see a rapidly alternating picture. It is created by the separate sets of symbols in the PRINT statements overlaying each other. At the same time, the GOTO statement in the last line creates a continuous loop: it tells the computer to go back to the start.

Without Lines 10 and 50 the program could not RUN properly. Both use the special characters available in the 'quote mode' of CBM machines, which allow you to incorporate cursor movements and other controls within an otherwise standard PRINT statement.

How do they work? The HOME symbol (reverse S) in Line 50 returns the cursor to the HOME position at the top left of the screen. This means that subsequent screen activity starts at this position, so the new characters from Line 60 onwards overwrite the existing ones.

The CLEAR/HOME symbol (reverse heart) in Line 10 does something more. Having returned the cursor to the top left position, it also clears the screen ready for the next image to appear.

SLOWING IT DOWN

Until now, the 'movement' of the insect is rather fast and could do with being slowed down. This is most easily done by using a FOR...NEXT loop. So next enter this line:

```
45 FOR T=1 TO 100: NEXT
```

When you press the RETURN key and RUN the program, the movement is much more deliberate. The FOR...NEXT loop is acting as a counter—in this case counting up to 100 before the program goes on to Line 50.

You can vary the length of the pause simply by changing '100' to any other number you choose: the bigger the number, the longer the delay.

Try also changing the position of the FOR...NEXT delay loop to Line 15. Depending on the delay you select, there should be a noticeable pause — a blank screen—when one image is cleared but not immediately replaced by another. This shows why it is better to use HOME rather than CLEAR/HOME, within a program of this kind.

As it stands, the image is still rather jerky, because the delay loop is acting only on the first image. You can get a much more purposeful movement by introducing another FOR...NEXT loop into the program to act on the second image. So enter this:

```
85 FOR I=1 TO 50: NEXT
```

This delay loop is shorter, to create a slightly irregular 'leg' movement, but you can of course change it if you wish.

MOVING THE CHARACTER

The next step is to alter the program so that the 'body' of the insect appears to cross the screen. Commodore BASIC has no PRINT AT statement, so other methods have to be used to position characters.

In simple applications, you can use the TAB function. TAB is always followed by either a number enclosed within brackets, such as TAB(15) — which places the cursor at column 15 on the screen — or by a variable enclosed within brackets. And TAB always forms part of the PRINT statement to which it applies.

In this case, use a variable to make the TAB position move across the screen. So use another FOR...NEXT loop:

```
10 FOR P=0 TO 35
20 PRINT "♥" TAB(P) ")))"
30 PRINT TAB(P) "OOO<"
40 PRINT TAB(P) ")))"
45 FOR I=1 TO 100:NEXT
60 PRINT "S" TAB(P) "((("
70 PRINT TAB(P) "OOO<"
80 PRINT TAB(P) "((("
85 FOR I=1 TO 50:NEXT
90 NEXT P
```

What you have done is to scrap the original GOTO statement in Line 90. It is now replaced by a FOR...NEXT loop which increases, by 1, the value of the variable P each time the program repeats itself. Since P is part of the TAB statement, this moves the insect across the screen by one step for every cycle of the program.

When you RUN the program, you can see the insect move across the screen and stop at the right. To re-start the action, you need:

```
100 GOTO 10
```



USING ROM GRAPHICS

The Commodore has a wide range of on-board graphics characters which can be used for creating rather more elaborate shapes and images.

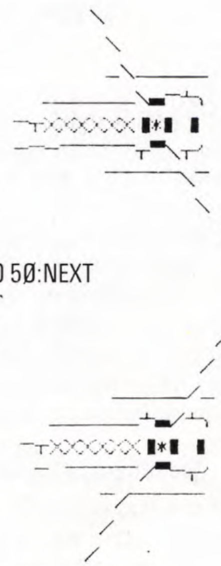
To access the full range you have to be familiar with the computer's "upper case and graphics" mode, obtained by simultaneously pressing the C= and SHIFT keys.

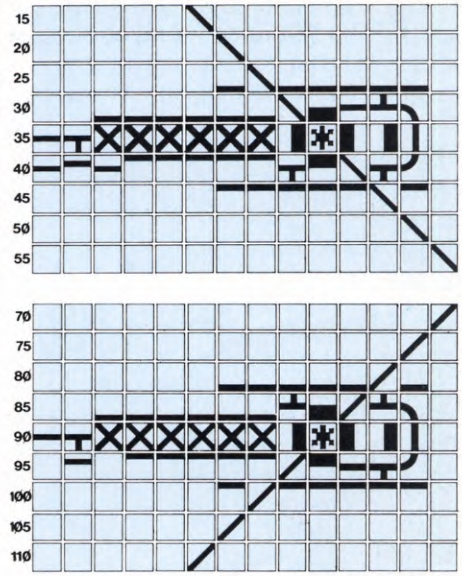
Now try keying in a few of the graphics. The left-hand symbol can be printed on the screen by simultaneously pressing C= and the chosen key letter. The right-hand symbol is obtained in the same way but by using the SHIFT and letter keys.

Other graphic symbols can be obtained by using RVS (reverse), enabled by simultaneously pressing CTRL and 9, and switched off by simultaneously pressing CTRL and Ø.

The helicopter graphic (fig 3) uses unreversed symbols, and gives some idea of ROM graphics. Here is the program:

```
5 PRINT "♥"
1Ø PRINT "▣"
15 PRINT □"                  "
2Ø PRINT □"                  "
25 PRINT □"                  "
3Ø PRINT □"                  "
35 PRINT □"                  "
4Ø PRINT □"                  "
45 PRINT □"                  "
5Ø PRINT □"                  "
55 PRINT □"                  "
6Ø FOR D=1 TO 5Ø:NEXT
65 PRINT "▣"
7Ø PRINT □"                  "
75 PRINT □"                  "
8Ø PRINT □"                  "
85 PRINT □"                  "
9Ø PRINT □"                  "
95 PRINT □"                  "
1ØØ  PRINT "                 "
1Ø5  PRINT "                 "
11Ø  PRINT "                 "
115 FOR D=1 TO 5Ø:NEXT
12Ø GOTO 1Ø
```





3. How to build a helicopter. Each square is a ROM graphic

Shown below is a small 'creepy crawly' which is easy to animate on the Spectrum or ZX81. To begin with, try creating him in a static position, thus:



```
1Ø PRINT AT 1Ø, 15; "OOO"
2Ø PRINT AT 9,15; ")))"
3Ø PRINT AT 11,15; ")))"
4Ø PRINT AT 1Ø,18; "<"
```

Now RUN the program. This is a rather elaborate way of producing a simple image, but it does illustrate one or two interesting points.

First, it gives you an idea of the relative positions of the screen locations. The middle of your insect is on Line 1Ø from the top, and about halfway across the screen.

Second, it shows the effect of telling the computer to print more than one character on a single location: it simply carries on and prints the extra characters on the adjoining locations. That is why the 'feelers' in Line 4Ø are at 1Ø,18. Locations 1Ø,15; 1Ø,16; and 1Ø,17; are already occupied by the insect's body.

A more convenient way of setting up the insect is to amalgamate the above instructions into a single line, thus:

```
1Ø PRINT AT 1Ø,15; "OOO<"; AT 9,15 ;")))";
   AT 11,15; ")))"
```

Add two more lines, and you have animation:

```
2Ø PRINT AT 1Ø,15; "OOO<"; AT 9,15; "(((";
   AT 11,15; "((("
3Ø GOTO 1Ø
```

When you RUN this program, you may find that it produces a blur — the pictures are being swapped too rapidly.



The best way to slow them down is to use a FOR...NEXT loop, which makes the computer count up to 1Ø (or whatever else you like) before PRINTing the next picture. So try adding these extra lines to your program. On the ZX81 use 2 instead of 1Ø.

```
15 FOR L=1 TO 1Ø
17 NEXT L
25 FOR M=1 TO 1Ø
27 NEXT M
```

Of course you can vary the length of the pause simply by changing the 1 TO 1Ø to 1 TO 5, 1 TO 2Ø or whatever else you need.

So far you have created a rather useless little insect, whose legs flail about madly but who doesn't move. That comes a little later (see **Movement**).



USING ROM GRAPHICS

Somewhat more interesting animation can be produced by using the standard Sinclair graphics characters. One example is in fig. 4.

The full program is given below, but if you are not used to the graphics symbols it will pay you to first create a static figure one line at a time, thus:

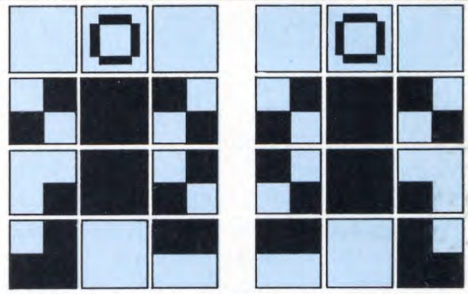
```
1 PRINT AT 5,15; "O"
2 PRINT AT 6,15; "▚█▞"
...and so on.
```

Getting these graphics characters is easy. To get those in Line 2 above, for example, press CAPS SHIFT (SHIFT on the ZX81) and 9 simultaneously. This puts you in graphics mode, indicated by a flashing G on the screen. Then, on the Spectrum, press CAPS SHIFT and 6 together to get an inverse – black for white – version of the symbol on the 6 key; then CAPS SHIFT and 8; then 6 by itself. On the ZX81, press SHIFT and T; SHIFT and SPACE; SHIFT and Y. Finally push 9 again to get out of graphics mode before inserting the quotes at the end.

Here is the full program for the figure:

```
1Ø PRINT AT 5, 14; "□O□"; AT 6, 14;
   "▚█▞"; AT 7., 14; "▗█▚";
   AT 8, 14; "▙□▀"
2Ø PRINT AT 5, 14; "□O□";AT 6, 14;
   "▚□▞"; AT 7, 14; "▚█▖";
   AT 8, 14; "▀□▛"
3Ø GOTO 1Ø
```

Again you may find that inserting a FOR ... NEXT loop after Line 1Ø, and again after Line 2Ø, is necessary to slow the action.



4. Dancer — in just 12 squares

MOVEMENT

Here, at last, is a program which not only animates the insect in the earlier program, but also makes him move across the screen:

```
1Ø FOR N=Ø TO 27
2Ø PRINT AT 1Ø,N;"OOO<"; AT 9,N;
   "((("; AT 11,N;"((("
3Ø PRINT AT 1Ø, N; "□□□□"; AT 9, N;
   "□□□"; AT 11, N; "□□□"
4Ø PRINT AT 1Ø, N; "OOO<"; AT 9, N; ")))";
   AT 11,N;")))"
5Ø PRINT AT 1Ø, N; "□□□□"; AT 9, N;
   "□□□"; AT 11,N;"□□□"
6Ø NEXT N
7Ø GOTO 1Ø
```

This program also uses a FOR...NEXT loop, but for a completely different purpose. In the example above, the computer was counting up fractions of a second before printing an image. Now, it is moving an image across the screen — one screen location (or 'box', if you like) at a time.

Why, then, does Line 1Ø read "Ø TO 27", when there are in fact 32 screen locations across a Sinclair screen? To find out, try replacing Line 1Ø in the program with:

```
1Ø FOR N=Ø TO 32
```

Another puzzle may be the need for Lines 3Ø and 5Ø. But try deleting them; you will soon see why they are needed.

NEXT STEPS

Now that you have started to learn animation and movement, you can begin to invent figures — or rockets or space ships — of your own, using the graphics set illustrated in your manual, the standard typewriter characters, or both.

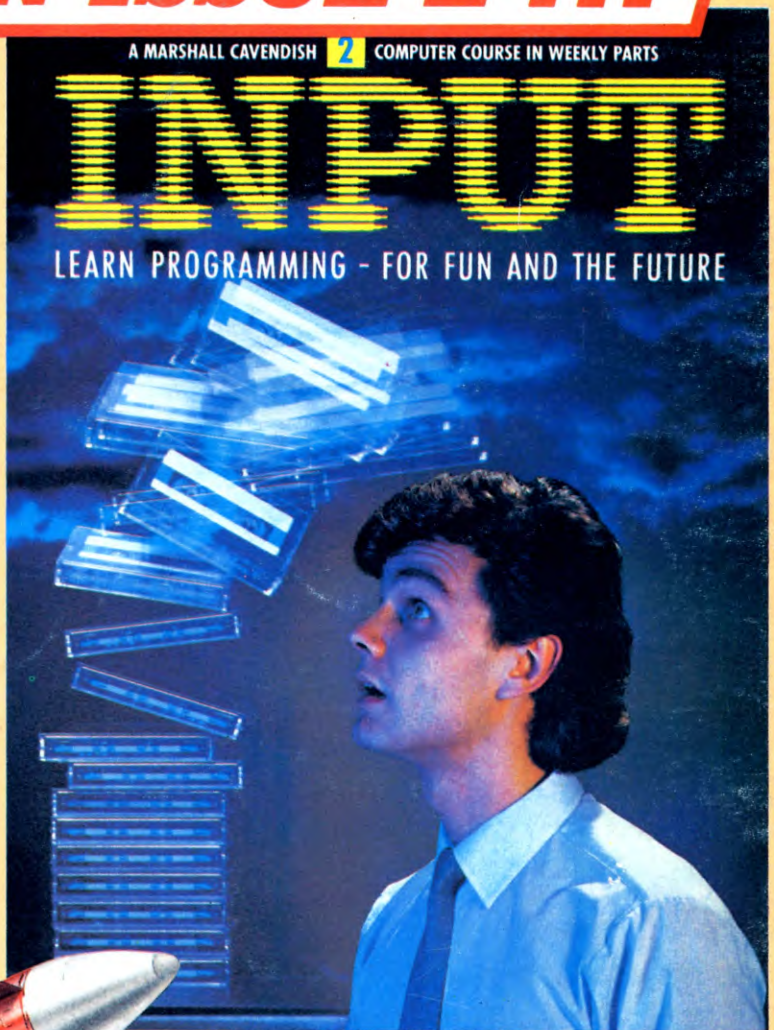
If you decide to try any of the figures illustrated for other makes of computer, however, you cannot use the programs as listed. Instead, you have to draw out the figures on graph paper and evolve your own programs from there.



CUMULATIVE INDEX

An interim index will be published each week. There will be a complete index in the last issue of *INPUT*.

COMING IN ISSUE 2 ...



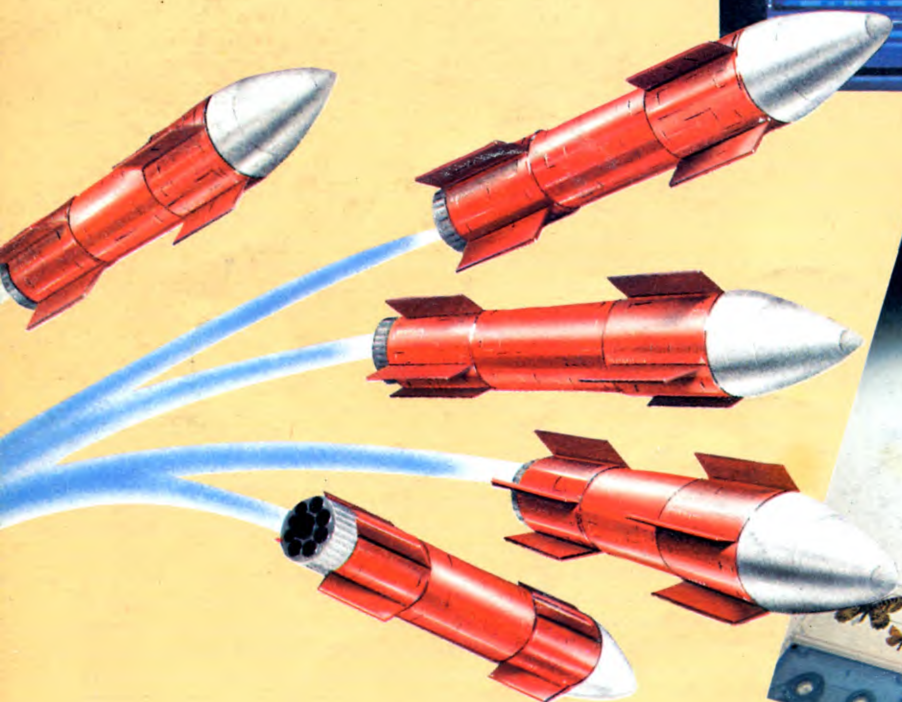
- *Get your games programs off the ground with* **KEYBOARD CONTROL** *techniques – how to make 'em move, how to make 'em fly and how to make 'em shoot.*

- *Fill the screen with graphic images with easy, short programs to create* **UDGs** *for games characters of your own design.*

- *Make your computer earn its keep with your own versatile* **DATA FILING** *system. You can store and recall information on hobbies, collections, names and addresses – and lots more . . .*

- *Learn how to use* **GOTO** *and* **GOSUB**, *the doors and gates that give your* **BASIC** *programs a logical structure.*

- *Put your computer's decision-making ability to work for you, with* **IF . . . THEN** *routines to add to your* **BASIC** *programming skills.*





ASK YOUR NEWSAGENT FOR INPUT